Towards a Novel Framework for Management of Context-aware Services and Networks

By
Nikolaos Vardalachos

Thesis submitted to the University of London for the degree of
Doctor of Philosophy in Electronic & Electrical Engineering

University College London
August 2005

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgements

There are many people I would like to thank for a huge variety of reasons. First of all, I would like to gratefully acknowledge the enthusiastic supervision of Professor Chris Todd and Alex Galis during this work. I could not have imagined having better advisors and mentors for my PhD, and without their common sense, knowledge, perceptiveness and cracking of the whip I would never have finished.

I would also like to express my thanks and appreciation to Prof. John Cosmas from the University of Brunel, as my external examiner, and Prof. Jonathan Pitts from Queen Mary, University of London, as my internal examiner, for their thoughtful questions at my defense.

I would like to thank Miguel Rio for his constructive comments during the preparation of this thesis. I am grateful to all my friends from the Network Services Group, University College of London, for being the surrogate family during the many years there.

Finally, I am forever indebted to my parents for their understanding, endless patience and encouragement when it was most required. I am also grateful to Maria for her support.

Nikolaos Vardalachos, August 2005

Abstract

There are very many different ways how context information could be used to make computer systems and applications more user-friendly, flexible, and adaptable. The use of context information is especially important in a mobile environment, where the environment of interaction, execution, and usage needs change rapidly. Areas where increased use of context information can bring added value and where research work is conducted include human computer interaction, adaptable user interfaces, virtual and augmented reality, mobile, ubiquitous, handheld, and wearable computing. The objectives of this thesis were to briefly clarify what context and context-awareness mean and to analyse the network context-awareness in networks through the use of policies. This was done through the development of a Context Policy Based Management Framework (Context Aware Policy Language and Policy Based Management Architecture), which was initially functionally tested at component level, and eventually successfully exercised by implementing two context aware services through the use of context aware policies.

Table of Contents

Table of Figures

Table of Tables

Acronyms and Abbreviations

3GPP	3rd Generation Partnership Project
AA	Active Applications
AC	Action Consumers
AN	Active Network
API	Application Program Protocol
ASN.1	Abstract Syntax Notation 1
ASP	Application Service Provider
ATM	Asynchronous Transfer Mode
CAPL	Context Aware Policy Language
CAPLDM	CAPL Decision Maker
CAS	Context Aware Service
CaSMIM	Connection and Service Management Information Model
CE	Condition Evaluator
CFP	Configuration, Fault and Performance
CIM	Common Information Model
CONTEXT	Active Creation, Delivery and Management of Efficient Context Aware Services
COPS	Common Open Policy Service
CORBA	Common Object Request Broker Architecture
CR	Conflict Resolver
CS	Context Service provider
DARPA	Defense Advanced Research Projects Agency
DB	Database
DEN	Directory Enabled Networking
DHCP	Dynamic Host Configuration Protocol
DINA	DINA Is Not ABLE
DMC	Decision Making Component
DMTF	Distributed Management Task Force
EMS	Element Management System
FCAPS	Fault, Configuration, Accounting, Performance, Security
FORCES	Forwarding and Control Elements Separation
GNMS	Generic Network Management System
GPRS	General Packet Radio Service
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
INMS	Inter-technology domain Network Management System

IP	Internet Protocol
LAN	Local Area Network
LDAPv3	Lightweight Directory Access Protocol version 3
LSP	Label Switch Protocol
LSR	Label Switch Router
MA	Mobile Agent
MIB	Management Information Base
MPLS	Multi Protocol Label Switching
MTNM	Multi-Technology Network Management
NE	Network Element
NMS	Network Management System
ODP	Open Distributed Processing
OMG	Object Managemnt Group
OSI	Open Systems Interconnection
PBM	Policy Based Managemnt
PBSM	Policy Based Service Management
PCIM	Policy Core Information Model
PCR	Policy Conflict Resolution
PDL	Policy Definition Language
PDL	Policy Definition Language
PDN	Policy Definition Notation
PDP	Policy Decision Point
PDU	Protocol Data Unit
PEP	Policy Enforcement Point
PFDL	Policy Framework Definition Language
PM	Policy Manager
PMT	Policy Management Tool
PN	Programmable Network
PSS	Policy Storage Service
QoS	Quality of Service
RFC	Request For Comments
SDH	Synchronous Digital Hierarchy
SL	Service Layer
SLA	Service Level Agreement
SMS	Service Management Systems
SN	Semantic Network
SNMP	Simple Network Management Protocol

SQL-	
RDBMS	SQL Relational DataBase Management System
TINA-C	Telecommunications Information Network Architecture Consortium
TMN	Telecommunications Management Network
VPN	Virtual Private Network
WBEM	Web-Based Enterprise Management
WDM	Wavelength Division Multiplexing
WiFi	Wireless Fidelity
WINMAN	WDM and IP Network MANagement
WLAN	Wireless Local Area Network
WMI	Windows Management Instrumentation
WO	WINMAN Operator
WWRF	Wireless World Research Forum
XML	eXtensible Markup Language

Introduction and Chapter Summary

When we humans interact with other persons and the surrounding environment we make use of implicit situational information. We can intuitively deduce and interpret the context of the current situation and react appropriately. For example, a person discussing with another person automatically observes the gestures and voice tone of the other party and reacts in an appropriate manner.

Computers are not as good as humans in deducing situational information from their environment and in using it in interactions. They cannot easily take advantage of such information in a transparent way, and if they can they usually require that it is explicitly provided. This is a challenge for human computer interaction. For example, present user interfaces still seldom can sense and adapt automatically to current light and noise level without the user providing the information. Another example comes from the area of mobile computing. Would it not be nice, if one could obtain services and information according to the current location and activity? For example, if at a stadium watching a football game, one could obtain additional information about the players, be able to participate in a local betting game, and check the traffic situation outside the stadium. There are very many different ways how context information could be used to make computer systems and applications more user-friendly, flexible, and adaptable. The use of context information is especially important in a mobile environment, where the environment of interaction, execution, and usage requirements change rapidly. Areas where increased use of context information can bring added value and where research work is conducted include human computer interaction, adaptable user interfaces, virtual and augmented reality, mobile, ubiquitous, handheld, and wearable computing.

The objectives of this thesis are to briefly clarify what context and context-awareness mean and to analyse context-awareness and more specifically network context-awareness in networks through the use of policies.

Chapter 1 describes different approaches in network and service management, starting from SNMP and eventually describing the Policy Based Network Management approach.

Chapter 2 describes what context and context awareness are, and tries to categorise and model context. Furthermore it explores the context awareness in overlay networks and the challenges involved and the possible application of policy-based management in context aware systems.

Chapter 3 describes the work carried out by the author in two European research projects (WINMAN and CONTEXT) during which he was involved in the design and implementation of policy based management systems. Both of these have offered hands on experience in the Policy Based Management field to the author and have influenced the development of the work described in this thesis.

Chapter 4 describes the Context Aware Policy Management Framework developed by the author in order to manage Context Aware Services and Networks. This chapter contains the main work carried out by the author and includes the development of a Context Aware Policy Language (CAPL) for expressing context aware policies, which would include context information, a Policy Management Tool (PMT) for authoring and managing the policies, a Policy Manager (CAPLPM) for handling those policies, a Decision Maker (CAPLDM) component for making decisions, a Policy Storage Service (PSS) for storing the policies and some Policy Enforcement Points.

Chapter 5 describes the enabling technologies. These include all the tools and devices used in order for the CAPL Framework to be developed. This includes Linux Routers (and their special functionalities), the use of an Active Platform (DINA), the use of WLANs and a SIP implementation (Siptrex).

Chapter 6 contains the functional testing of each of the components developed. A set of functional tests were developed for all the CAPL Framework components together with their expected results.

Chapter 7 contains the description and implementation of two services designed by the author, in order to be able to exercise the previously designed framework. The two services developed were the FollowMe Service, which allows the subscribed user to connect to his enterprise network wherever he is, and the Emergency Support Service, which during a crisis can block all but the privileged users, from making use of the SIP telephony service.

Chapter 8 contains the discussion section of the thesis. It summarises all the work carried out by the author and also investigates possible improvements and extensions to it.
Appendices A.1 - A.8 include work carried out by the author, which cannot fit within the current structure of this thesis, but is useful in order for the reader to comprehend the work described in the main part of the thesis. Furthermore, a more detailed description of the CONTEXT policy management system is given.

1 Network and Service Management

1.1 Introduction

Network management stems from the realisation that hosts, routers, and other networking devices often require maintenance operations, and the network is a communications medium, so why not use the network to perform the maintenance? It is concerned with; running the system, optimising the usage of the system (optimisation/cost control) and with providing services to business customers and end users. The overall cost of network management may be up to 30% of the installation costs of a network. The cost of poor network management can even be higher [1]. Poor utilisation of the network can be expensive [2] and when considerations of customer care in free market environments, are taken into account, penalties through lost custom can be high.

The oldest and simplest form of network management is the remote login. In fact, most fancy routers support TELNET access to some sort of command prompt. Many operations can be performed in no other way. However, more sophisticated network management tools have been developed, for a variety of reasons. Remote logins are designed for human interaction, and use command style and syntax that vary between different hardware and software platforms. A more specialized and standardized approach allows automated software tools to easily perform management operations on a variety of platforms. Also useful is a standard method of reporting network failures and error conditions to a centralised location.

The central focus of much international research and commercial development has been the Telecommunications Management Network (TMN) standards. FCAPS (fault-management, configuration, accounting, performance, and security) is an acronym for a categorical model of the working objectives of network management. There are five levels, called:

- Fault-management level (F),
- Configuration level (C),
- Accounting level (A),

- Performance level (P),
- Security level (S)

At the F level, network problems are found and corrected. Potential future problems are identified, and steps are taken to prevent them from occurring or recurring. Hence, this includes:

- Handling of alarms: prioritising condensing and filtering
- Fault analysis – immediate & long term
- Maintenance dispatch

In this way, the network is kept operational, and downtime is minimised.

At the C level, the network operation is monitored and controlled. Hardware and programming changes, including the addition of new equipment and programs, modification of existing systems, and removal of obsolete systems and programs, are coordinated. An inventory of equipment and programs is kept and updated regularly. Hence, this includes:

- Identity, status & location of equipment
- Routing, switching tables
- Connection tables
- Directory maintenance
- Network intelligence maintenance

The A level, which might also be called the allocation level, is devoted to distributing resources optimally and fairly among network subscribers. This makes the most effective use of the systems available, minimising the cost of operation. This level is also responsible for ensuring that users are billed appropriately. Hence, this includes:

- Collecting bills / usage information
- Issuing bills
- Analysis of usage (sales)
- Forecast & Capital Expenditure

The P level is involved with managing the overall performance of the network. Throughput is maximized, bottlenecks are avoided, and potential problems are identified. A major part of the effort is to identify which improvements will yield the greatest overall performance enhancement. Hence, this includes:

- Collecting of traffic information
- Traffic flow analysis
- Traffic flow prediction (by hour, day, month , year)

At the S level, the network is protected against unauthorised users, and physical or electronic sabotage. Confidentiality of user information is maintained where necessary or warranted. The security systems also allow network administrators to control what each individual can have access to. Hence, this includes:

- Encryption (secrecy), data integrity, behavioural integrity
- Non-repudiation
- Key management
- Access control

The users' participation generates high priorities at traditionally the accounts, fault management and of course, the network usage processes. Furthermore, developing demands – especially in the corporate level- require user interaction with configuration management systems for personalised & direct control of private virtual networks; and with performance management for individual feedback of status and usage.

1.2 Simple Network Management Protocol

Currently the most popular approach for network management is the SNMP Protocol. The *Simple Network Management Protocol (SNMP)* is an application layer protocol that facilitates the exchange of management information between network devices. It is part of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth.

Two versions of SNMP exist: SNMP version 1 (SNMPv1) and SNMP version 2 (SNMPv2). Both versions have a number of features in common, but SNMPv2 offers enhancements, such as additional protocol operations. Standardisation of yet another version of SNMP—SNMP Version 3 (SNMPv3)—is pending. Figure 1 illustrates a basic network, managed by SNMP.

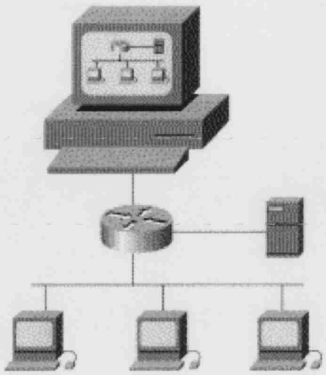


Figure 1: Simple SNMP managed Network

1.2.1 SNMP Basic Components

An SNMP-managed network consists of three key components (Figure 2): managed devices, agents, and network-management systems (NMSs).

- A *managed device* is a network node that contains an SNMP agent and that resides on a managed network. Managed devices collect and store management information and make this information available to NMSs using SNMP. Managed devices, sometimes called network elements, can be routers and access servers, switches and bridges, hubs, computer hosts, or printers.
- An *agent* is a network-management software module that resides in a managed device. An agent has local knowledge of management information and translates that information into a form compatible with SNMP.
- An *NMS* executes applications that monitor and control managed devices. NMSs provide the bulk of the processing and memory resources required for network management. One or more NMSs must exist on any managed network.

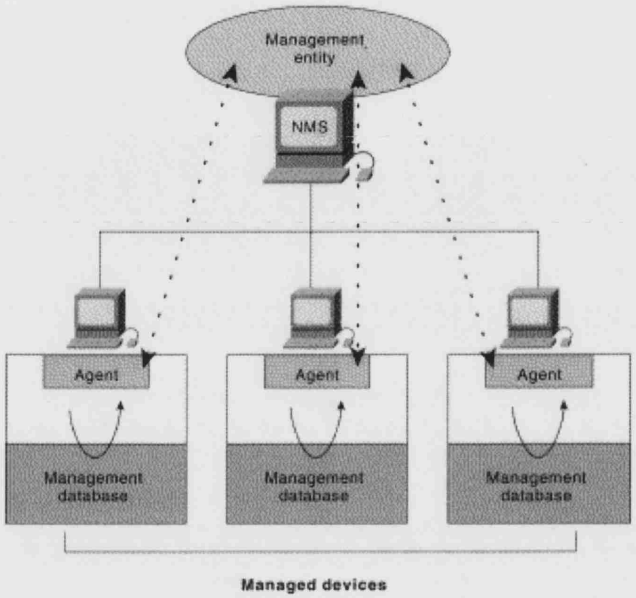


Figure 2: The main SNMP entities

1.2.2 SNMP Overview

The Simple Network Management Protocol (SNMP) is essentially a request-reply protocol running over UDP (ports 161 and 162), though TCP operation is possible. SNMP is an asymmetric protocol, operating between a management station (smart) and an agent (dumb). The agent is the device being managed - all its software has to do is implement a few simple packet types and a generic get-or-set function on its Management Information Base (MIB) variables. The management station presents the user interface. Simple management stations can be built with UNIX command-line utilities. More complex (and expensive) ones collect MIB data over time and use Graphical User Interfaces (GUIs) to draw network maps.

SNMP Version 1 is documented in RFC 1157. SNMP Version 2 is documented in several RFCs:

- RFC 1902 (MIB structure) [35]
- RFC 1903 (Textual Conventions) [36]
- RFC 1904 (Conformance Statements) [37]

- RFC 1905 (Protocol Operations) [38]
- RFC 1906 (Transport Mappings) [39]
- RFC 1907 (MIB) [40]

SNMP's packet formats are described using *Abstract Syntax Notation 1 (ASN.1) [41], [76], [77]*, one of ISO's "Open" protocols. ASN.1, like all OSI standard documents, is not freely available on-line. An SNMP operation takes the form of a Protocol Data Unit (PDU), basically a fancy word for packet. Version 1 SNMP supports five possible PDUs:

- **GetRequest / SetRequest** supplies a list of objects and, possibly, values they are to be set to (SetRequest). In either case, the agent returns a GetResponse.
- **GetResponse** informs the management station of the results of a GetRequest or SetRequest by returning an error indication and a list of variable/value bindings.
- **GetNextRequest** is used to perform table transversal, and in other cases where the management station does not know the exact MIB name of the object it desires. GetNextRequest does not require an exact name to be specified; if no object exists of the specified name, the next object in the MIB is returned. Note that to support this, MIBs must be strictly ordered sets (and are).

Trap is the only PDU sent by an agent on its own initiative. It is used to notify the management station of an unusual event that may demand further attention (like a link going down). In version 2, traps are named in MIB space. Newer MIBs specify management objects that control how traps are sent.

1.2.3 Issues with SNMP

Although SNMP is still very much in use, it is not the panacea for all network management problems. This stems from the fact that the administrator in order to deploy a service, would need to know the type of devices he needs to manage and then use the appropriate MIBs[1] in order to achieve his goals. Also, another key challenge is to manage the network resources efficiently as a whole in order to set up a service. For this reason, a

[1] SNMP lacks standardized MIB objects usable for configuration [136]

high level view of the network and its resources is required to be used. Different approaches for network management have been developed through the years; such include approaches as the TMN, ODP, TINA-C, WBEM and Policy Based Management. The telecommunications management network (TMN) provides a framework for achieving interconnectivity and communication across heterogeneous operating systems and telecommunications networks. TMN was developed by the International Telecommunications Union (ITU) [7] as an infrastructure to support management and deployment of dynamic telecommunications services. The Open Distributed Processing Framework Reference Model (ODP-RM) [8] was developed by the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC) and ITU-T for distributed processing. It goes one step further than the TMN approach, enabling the design of management applications that are independent of distribution, the underlying infrastructure and the management protocols [81]. The Telecommunications Information Networking Architecture Consortium (TINA-C) Framework was developed by the TINA-C consortium [9], [10], and its purpose was to ensure interoperability, portability and reusability of software components and independence from specific technologies, and to share the burden of creating and managing a complex system among different business stakeholders, such as consumers, service providers, and connectivity providers. TINA provides for two major separations of concern; the first separation is between applications and the environment on which they run. The second is separation of applications into the service-specific part and the generic management and control part. TINA success stories can be found at [82], [83]. Web-Based Enterprise Management (WBEM) [11], [12], known as Windows Management Instrumentation (WMI) provides tools and a single management methodology for a wide range of problems [84], [85], [86], [87]. It does not attempt to replace existing standards and technologies as SNMP and CMIP but complements these initiatives, providing a layer from which data can be accessed with a common integrated view.

The Policy Based Network Management (PBNM) has been a 'hot' research topic in the last period. The IETF has been working in this area, through two of its groups, in order to provide a means for managing IP based networks, offering some QoS guarantees (Policy

Based Networking). The aim of the policy-based management is to apply integrated management system so that system management, network management, and application management can cooperate. The Policy Based Management approach offers many benefits making it more attractive as a solution to Network Management. These include flexibility, adaptability, uniqueness, specificity etc [137]. The Policy Based Management architecture will be described in the next section.

1.3 Policy Based Management

Policy Based Management (PBM) is a relatively new field to the Internet community, but has already been promoted by several network equipment vendors. The current goal of PBM is to provide facilities, which allow control of multiple types of devices that must work in concert to provide a desired service. Examples of devices, which can be "PBM-enabled", are hosts (clients and servers), routers, switches, firewalls, bandwidth brokers, sublet bandwidth managers and network access servers. Examples of services controlled by PBM can be Quality of Services (QoS) reservations, Virtual Private Networks (VPN), etc.

For the deployment of Policy Based Management Systems in the Internet, a standardisation process is required, to ensure the interoperability between equipment from different vendors and PBM systems from different developers. Both the Internet Engineering Task Force (IETF) [42] and the Distributed Management Task Force (DMTF) [43] are currently working for the definition of standards for Policy Based Network Management. The DMTF is mainly focused on the representation of policies and the specification of a corresponding information model and schema. The IETF is also working in that field, in co-operation with DMTF, but also trying to define a general framework for a PBM system, as well as a protocol that could be used for implementing a PBM system.

The most important contribution of DMTF is that it has defined the Common Information Model (CIM) [44] management schema, which consists of an object-oriented model for

the representation of the information that will be stored in the directory of a DEN[2]-enabled network [78]. But the CIM, is an information model for general management information and does not yet have a direct relationship with policy based network management. DMTF is also working in association with the IETF policy workgroup, to extend the CIM syntax and make the CIM usable in real PBM systems. Policy Core Information Model Extensions [45] has also been defined by IETF, which means that IETF policy architecture has covered almost all the components required by a PBM system.

The IETF policy workgroup has specified the following components for policy based network management architecture:

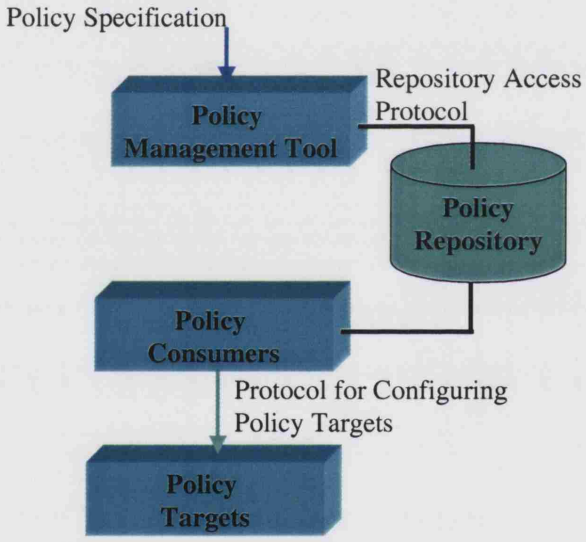


Figure 3: Basic PBM Architecture given by IETF

The IETF PBM architecture, as shown in Figure 3, includes the following four functional components:

- Policy Management Tool

  With this tool, new policies for the system can be defined, existing ones edited, or simply viewed[3]. The definition of policies may be done in a higher-level language,

[2] Directory-Enabled Networking (DEN) is an industry-standard initiative and specification for how to construct and store information about a network's users, applications, and data in a central directory.

[3] These policies exist in the policy repository, and therefore in the network.

which offers abstraction. In this case, the tool also performs a translation of these high level policies to a set of policy rules that can be interpreted by the policy consumer. The information model and schema, also defined by the IETF, specify the format of the policy rules.

- Policy Repository

  The policy repository is used for the storage of policies, after they have been defined and validated by the policy management tool. There should also exist a protocol that enables read and write access to the repository. The general framework does not require a specific implementation for the policy repository, or the repository access protocol. However, the current work of the IETF includes the use of a directory server, with the Lightweight Directory Access Protocol version 3 (LDAPv3) [79] for access. The policies are stored in a format compliant with the CIM (Common Information Standard) specification by the DMTF. Structured Query Language - Relational database management system (SQL-RDBMS) [80] is also a widely used option.

- Policy Consumer or Policy Decision Point (PDP)

  The policy consumer is the component that retrieves policies from repository, evaluates them and sends the necessary commands to the policy target. Of course the evaluation of certain policies may also be done on the policy target, if it is capable of understanding policies. The distribution of the policy evaluation process between the consumer and the target also depends on the conditions of a policy. For example if the target is policy-aware and the policy condition concerns only the specific device, the evaluation can be done on the policy target. If however the condition is time-based or depends on the overall state of the network, then the evaluation should be done by the consumer. Additionally, the policy consumer performs a local conflict check, checking only those devices that are controlled by the specific consumer. The policy consumer also checks if the resources needed for a specific policy are available in all the controlled devices. A policy consumer can be integrated into a policy target, or it may exist in another device. However there is a need for at least one consumer in every administrative domain. The Policy Consumer is usually called *Policy Decision Point (PDP).*

- Policy Target or Policy Enforcement Point (PEP)

  The policy target is the managed device, where the policy is finally enforced. In PBM system, Policy Target often means *Policy Enforcement Point (PEP)*. There is a need for a transport protocol for communication between the policy consumer and the policy target, so that the consumer can send policy rules or configuration information to the target, or read configuration and state information from the device. There is not a requirement for a specific protocol for this operation. However, the Common Open Policy Service (COPS) [46] protocol, defined by the IETF Resource Allocation Protocol workgroup, is becoming the standard.

Please note that PDP and PEP can reside either on physically different machines or on the same machine.

1.4 Policy Languages

Policy Definition Language (PDL) is designed for satisfying the need of mapping requirements for services to be provided by the network as defined in a business specification (e.g., an SLA[4]) to a common vendor- and device-independent intermediate form. This enables policy information and specifications to be shared among the heterogeneous components that comprise the policy framework, and allows multiple vendors to use multiple devices to implement that framework. From the administrator's point of view, PDL is used to define new policies in terms of policy rules with condition and action lists. In the following sections, several PDLs will be presented.

1.4.1 Policy Framework Definition Language (IETF)

The Policy Framework Definition Language (PFDL) proposed by IETF was supposed to satisfy the objectives of a PDL as mentioned in the very beginning of this sub-section. But due to some unclear reason, the IETF policy group has decided to suspend work on

[4] Service Level Agreement

the policy definition language. The November 1998 draft [47] outlines a possible policy rule language to fit into the IETF framework. This draft mainly describes the structure of policies in the PFDL, which also covers policy component hierarchy, policy conflict detection, service and usage policies, and collective aspects of policy (i.e. roles and groups). In [48] it gives an example of such a policy:

If any of the named users below tries to listen to real audio streams during working hours, then deny the service.

```
IF USER IN {"peter", "frank", "mary"} AND
      TIME IN Mon - Fri AND
      TIME IN 09:00 - 17:00 AND
      HTTP_URL_PATH ENDS ".ram"
THEN
      DENY
```

Unfortunately, this early draft was said to be neither fully consistent nor did it provide enough details to answer most of the immediately arising questions related to it such as the formal expression of its syntax and semantics. Furthermore, no compiler was discussed in the PFDL work.

1.4.2 Ponder

Ponder [22],[23] is a language developed by the Policy Research Group in Imperial College London [49] for specifying management and security policies for distributed systems and network management systems. Ponder supports authorisation, filter, refrain and delegation policies for specifying access control and obligation policies to specify management actions. Furthermore, Ponder provides a uniform means of specifying policies relating to a wide range of management applications – network, storage, systems, application and service management. In addition, it supports a common means of specifying enterprise-wide security policies that can then be translated onto various security implementation mechanisms. Ponder is declarative, strongly-typed and object-oriented which makes the language flexible, extensible and adaptable to a wide range of management requirements [50]. As part of the Ponder development effort, a complete toolkit has been developed to support the users of the language. Ponder is a proprietary language/platform that is more concerned about security policies.

1.4.3 Other Approaches

Apart from the aforementioned efforts on defining policy languages, work in this field has been carried out by others. Efforts worth mentioning are the following:

- Policy Definition Notation (PDN) [51]

  This approach is based on the ODP distributed computing platform, and the authors choose the area of performance management for the application of policies. Their notation looks very much like a programming language[5].

- Policy Template Definition [52], [53]

  In this approach, apart from the policy classification and the policy hierarchy, R. Weis also derived a policy template definition, which is then transformed into policy objects/management scripts.

- Policy Definition Language (PDL) [54]

  This approach defines a three level policy hierarchy. This allows a stepwise refinement of policies from an informal strategic level or requirement level then via a Goal-oriented level to a formalised operational level. Different language constructs are used to specify policies for each abstraction level. This work is influenced by the early proposals for policy notation by the Imperial College Group [55].

1.5 Policy Core Information Model

The IETF Policy Core Information Model (PCIM) presents the object-oriented information model for representing policy information developed jointly in the IETF Policy Framework Working Group and as extensions to the Common Information Model (CIM) activity in the Distributed Management Task Force (DMTF). This model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and association classes that indicate how instances of the structural

[5] This makes the use of it, difficult for users not familiar with it.

classes are related to each other. The mappings of this information model to various concrete implementations, for example, to a directory that uses LDAPv3 as its access protocol will be discussed in the next section.

The classes comprising the PCIM (and its extensions) are intended to serve as an extensible class hierarchy (through specialisation[6]) for defining policy objects that enable application developers, network administrators, and policy administrators to represent policies of different types. The major objective of information models is to bridge the gap between the human policy administrator who inputs the policies and the actual enforcement commands executed at the network elements.

1.6 Policy Storage Schema

The intention of policy storage is to provide a logically single and centralised repository that stores the relationship of users and applications and their relationship to network management/services, in order to enable management interoperability[7] and information sharing. This storage schema should be fully aligned with the policy information model, in order to provide appropriate mapping between the entities managed and the ones stored. The commonly used way for policy storage is directories following ISO X.500 [120]. The directory objects are expected to represent networks/subnets, services, devices, applications, users or locations. The relationships of these objects are also expected to be defined and managed in the directory. The management data can be accessed through ties to the directory objects. In this way, a single model is available for the access to the objects, enabling the interoperability of different management systems.

.

1.6.1 Directory Service

The term *directory service* means the collection of software, hardware, and processes that store information that is needed and ensures that this information is available to users or to applications. A directory service consists of at least one instance of *Directory Server*

[6] This can be done by specialising the abstract classes to fit the required entities to be managed

[7] The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together.

and one or more *directory client* programs. One common directory service example is a Domain Name System (DNS) server [121]. A Directory Server stores all of the information in a single, network-accessible repository. Directory Server serves the needs of a wide variety of applications. Therefore, it requires a network-based standard means of communicating between the applications and the directory. That is where LDAP (Lightweight Directory Access Protocol) [130] comes into place. LDAP gives applications access to the global directory service provided by directory server [131].

LDAP provides a common means for client applications and server to communicate with each other. LDAP is a "lightweight" version of the Directory Access Protocol (DAP) used by the ISO X.500 standard. DAP gives any application access to the directory via an extensible and robust information framework, but at a high administrative cost. DAP uses a communications layer (not the Internet standard TCP/IP protocol) and has complicated directory-naming conventions. LDAP preserves the best features of DAP while reducing administrative costs. LDAP uses an open directory access protocol running over TCP/IP and uses simplified encoding methods. It retains the X.500 standard data model and can support millions of entries for a modest investment in hardware and network infrastructure.

The Directory Server includes the directory itself, the server-side software that implements the LDAP protocol, and a graphical user interface that allows end-users to search and change entries in the directory. Also, one can write his own client software using the LDAP client Software Development Kit (SDK) usually provided by the directory server product in order to access the server. Multiple client programs can speak to the server using LDAP. Please note that directory data can be physically located on different machines across the network. Most of the directory products support this functionality.

1.6.2 Directory Schema

The directory schema is a set of rules that defines how the data can be stored in the directory. The data is stored in the form of directory entries. Each entry is a set of attributes and their values. Each entry must have an object class. The object class specifies the kind of object the entry describes and defines the set of attributes it contains.

The schema defines the type of entries allowed, their attribute structure and the syntax of the attributes. The schema can be modified and extended if it does not meet your required needs.

In LDAP, an object class defines the set of attributes that can be used to define an entry. The LDAP standard [122], [123], [124], [125], [126], [127], [128], [129] provides some basic types of object classes, including: Groups (including unordered lists of individual objects or groups of objects), Locations (such as the country name and description), Organizations, People, and Devices.

1.6.3 IETF Policy Core LDAP Schema

Within the IETF policy framework, IETF recommends that Policy Core LDAP Schema (PCLS) or a schema that subclasses PCLS[8] shall be used for PCIM-based policy storage. Basically, PCLS defines a mapping of PCIM to a form that can be implemented in a directory that uses Lightweight Directory Access Protocol (LDAP) as its access protocol [116]. Two types of mappings are involved:

- For the structural classes in the information model, the mapping is basically one-for-one: PCIM classes map to LDAP classes, information model properties map to LDAP attributes.
- For the relationship/association classes in the information model, different mappings are possible. As proposed by PCLS, the PCIM's association classes and their properties are mapped in three ways: to LDAP auxiliary classes, to attributes representing distinguished name (DN) references [132], and to superior-subordinate relationships in the Directory Information Tree (DIT) [133].

PCLS is closely aligned with the work being done in the DMTF which has adopted the Directory Enabled Network (DEN) [117], [118] specification as policy storage

[8] This would extend the current PCLS classes

mechanism. DEN is also used to map network management/services to a directory that follows CIM[9].

1.7 Conclusions & Summary

This chapter presented the state of the art in the Policy Based Management field, which will be used as the basis for the main part of the work described in chapter 4. Before describing the Policy Based Management approach in detail, a short history review was carried out, by describing the Simple Network Management Protocol, its components and also some of its shortcomings. Then, the Policy Based Management's main scope and functional components were described; this included the Policy Management Tool, the Policy Repository, the Policy Decision Point and the Policy Enforcement Point. Also, a short description of different approaches to create Policy Definition Languages was carried out. Finally, the Policy Core Information Model was described, which will be used as the basis for the specification and development of the CAPL language developed by the author in chapter 4. Policy Based Management offers a more flexible and customisable management solution, allowing for on the fly configuration of devices. As such, it is suitable for the management of large-scaled networks. The use of policies and hence Policy Based Management may soon become less of an option and more of a necessity as the number and types of network elements increase.

The next chapter will describe the concepts of Context and Context awareness. Context awareness may allow for the services offered to become more flexible and more autonomous, in order to satisfy the growing and changing requirements of the users. The combination of chapter 1 and 2 define the main aim of this thesis, which is to offer a way of managing context awareness through the use of policies.

[9] Please refer to [117]for more information on DEN and its LDAP schema for CIM and to [116] for PCLS classes description.

2 Context and Context Awareness

2.1 Introduction

As computer and network become more pervasive (therefore computing is getting more pervasive), the nature of services should change accordingly. Services must become more flexible in order to respond to highly dynamic computing environments, and become more autonomous to satisfy the growing and changing requirements of users. That is, services must become more context-aware, or context-aware service needs to be provided.

This section is a review of the state-of-the-art in context-awareness and context categorisation and finally discusses a way of managing context awareness through the use of policies.

2.2 What is Context?

Context, as a concept, has emerged from a group of interrelated areas of research. It started from the vision of mobile computing going through ubiquitous and pervasive computing and reaching ambient computing. There is a great variety of papers and publications that propose, under different points of view, what should be understood as context[10]. Many of them try to define context giving different examples of types of information that the authors believe should form part of the definition, like the location of the service's user or the physical characteristics of his surroundings. In each of these papers, the importance of context has been proposed for enhancing human-computer interaction, thereby providing seamless computing all the time and everywhere. There seems to be no formal accepted definition for context. This can lead to the misunderstanding of the term 'context' in many cases.

Context as Dey et al. in [26] states:

[10] See following references [26], [29], [30], [31], [32]

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."

Almost any information available at the time of an interaction can be seen as context information. Some examples are:

- *identity*
- *spatial information - e.g. location, orientation, speed, and acceleration*
- *temporal information - e.g. time of the day, date, and season of the year*
- *environmental information - e.g. temperature, air quality, and light or noise level*
- *social situation - e.g. who you are with, and people that are nearby*
- *resources that are nearby - e.g. accessible devices, and hosts*
- *availability of resources - e.g. battery, display, network, and bandwidth*
- *physiological measurements - e.g. blood pressure, heart rate, respiration rate, muscle activity, and tone of voice*
- *activity - e.g. talking, reading, walking, and running*
- *schedules and agendas*

Schilit and Theimer [29] refer to context as *location, identities* of nearby people and objects and changes to those objects.

Brown *et al.* [30]define context as *location, identities* of the people around the user, the *time* of day, season, temperature, *etc.* Context is seen to be the elements of the user's environment that the user's computer knows about.

Schmidt, *et al.* [31], [32] describes context as the knowledge about the user's and device's state, including *surroundings, situation*, and to a less extent, *location.* This definition of context requires understanding of the exact meaning of the words *state* and *surroundings*, and it was the first step taken towards understanding and defining context.

Dey's definition is both strong and comprehensive, but there are some context issues which appear not to be included but raise contradictory issues. On one hand, the scope of context can change from situation to situation. The author believes that this definition is too generic in terms of the scope of entity (any object that relates) to provide any clear clue as to how to select relevant context when building a context-aware system; on the

other hand, it shouldn't limit the entities to just those that only exist between a user and an application. Day implies context is constrained within information existing only between user and application. This indicates the origination of context (and context-awareness) to the Human-Computer Interface (HCI) research field. This thesis takes one step further to extend the scope of context research into networks (both wired and wireless networks) by considering network context-awareness, especially network centric context-aware services. This will be further discussed in the following sections regarding context awareness and context categorisation.

2.3 Context-Awareness

Context-awareness means that one is able to use context information. A system is context-aware if it can extract, interpret and use context information and adapt its functionality to the current context of use. The challenge for such systems lies in the complexity of capturing, representing and processing contextual data [26]. To capture context information generally some additional sensors and/or programs are required. To transfer the context information to applications and for different applications to be able to use the same context information, a common representation format for such information should exist. In addition to being able to obtain the context-information, applications must include some "intelligence" to process the information and to deduce the meaning. This is probably the most challenging issue, since context is often indirect or may only be deducible by combining different pieces of context information. E.g. if three persons meet in a certain office room at a certain time, it may mean that it is the weekly strategy meeting.

There are features that are characteristic for context-aware applications [26], [27], [28]:

- *Firstly, information and services can be presented to the user according to the current context.* This includes the selection of proximate information and services, and contextual commands. An example of the former would be information on where the closest bank is. An example of the latter would be a user interface changing commands depending on the time of the day or location.

- *Secondly, automatic execution of a service when in a certain context.* This includes context-triggered actions and contextual adaptation. An example of the former would be that when a user enters a specific room his mails would be shown on a nearby terminal. An example of the latter would be the change of volume on a phone according to the current noise level.
- *Thirdly, tagging of context information, for later retrieval; this is self explanatory.*

References [28], [32] and [34] include very helpful general discussions about context-awareness and its challenges.

2.4 Categorising Context

On top of these definitions, researchers started categorising context as a technique for separating and clarifying the context's boundaries in different situations.

Tuulari [33] divides context awareness into two parts: self-contained context awareness and infrastructure based context awareness. The former implies context awareness achieved without any outside support and the latter implies context awareness achieved with outside support.

Chen and Kotz [34]extended Schilit's division of context into four categories to achieve a better understanding of the concept.

Computing context includes network connectivity, bandwidth, communication costs, and nearby resources such as printers, displays, and workstations.

User context includes user profiles, user location, and nearby users and people.

Physical context includes lighting, temperature, and humidity.

Time context includes time of the day, week, year, and also the season of the year.

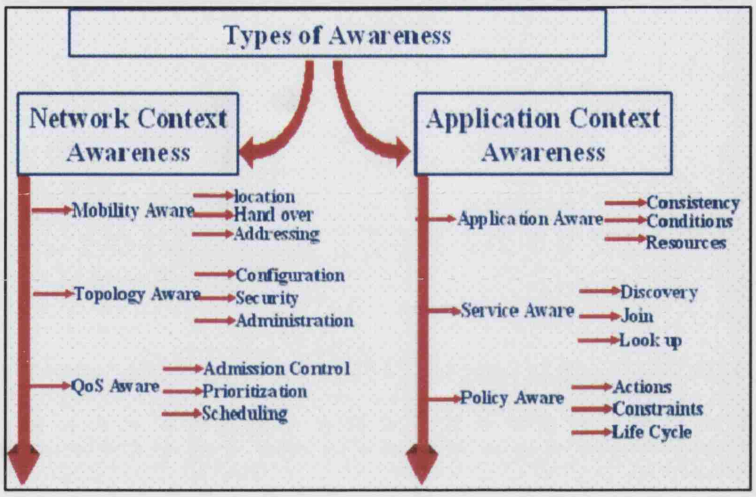


Figure 4: Categorisation of context aware attributes

This context categorisation has the advantage over other classifications in the methodology used for classifications, which is based on the entities using context as input information. From this, the author has synthesised a novel categorisation of Context awareness based on the granularity of context information which is composed by two categories, an application-related and network-related category (Figure 4). Each category has its own sub categories that further specialises context in its domain. In the application-related category, the application aware subcategory has *consistency*, used resources and *conditions* as the core functionalities that use context information to facilitate the process of inter, and intra application interactions. The service aware category includes *discovery, join*, and *lookup processes* for sighting services matching the required services that are currently needed. The other root category is the network aware category with subcategories such as mobility-aware - an essential component in context aware systems with functionalities such as hand over, location detection, and auto configuration according to current location. This category of context is of great interest and it will be the focus of the work carried out by the author in chapter 4. Mobility awareness is important in ad-hoc networks (section 2.5). Other categories include topology aware and QoS aware.

2.5 Context –Awareness in Ad-hoc Networks

An Ad-hoc network [101], [102] is a network, which is established dynamically by (mobile) devices and is maintained by them for their communication needs. The devices themselves act as network nodes routing the traffic. No additional infrastructure is required. Since the nodes can come and go the network topology may change frequently. The networks may operate in isolation or may have gateways and interfaces with the fixed network.

Spontaneity, dynamics, and mobility characterise Ad-hoc networks. In an environment of such a nature, it is likely that context-awareness can bring added value. Actually context information can be used on several different logical levels in Ad-hoc networks: in establishing an ad-hoc network, in the routing mechanisms, and on application level.

2.5.1 Establishing an Ad-hoc Network

Ad-hoc networks are generally intended to be established between devices situated in a certain logical or physical location. To find other network nodes different methods can be used, e.g. message broadcasting, lookup of devices using the same base station or gateway, etc. One could state that Ad-hoc networks are context-aware per se because of this fact.

In addition to using device location and proximity, almost any other type of context information could be used to determine which devices should establish a network. This includes e.g. identity information, friends nearby, time of the day, etc. For example, an Ad-hoc network could be established when certain friends are close nearby, or only during a certain time of the day.

2.5.2 Routing Mechanisms

The Ad-hoc routing mechanisms could actually take advantage of context information to a broader extent. Parameters that could be used are location, physical proximity, device identity, liability of a node, different characteristics of a node, e.g. bandwidth capacity,

owner of the node, cost to use the node, etc. However, the downside is that routing optimisation can get complicated if too many parameters are used at a time.

2.5.3 Applications

Many scenarios have been discussed where mobile device users (laptop, pda, and mobile phone) establish Ad-hoc networks at public places such as airports, convention centres, and similar places. Another application area widely discussed is the conferencing/meeting room Ad-hoc networking. What kind of context-aware applications would then be the most appropriate ones for Ad-hoc networking and how could they take advantage of the context information?

A very evident type of application is the ability to find nearby resources. This can include input and output devices: printers, displays, speakers, facsimiles, video cameras, thermostats, etc. In pure service lookup actually even location information does not need to be used. In situations where distance calculations have to be done location information of the client is required.

Another type of applications suitable for Ad-hoc networks can be derived from the idea of being able to create local network communities, not only based on the persons available at a certain site but also based on additional information. For example, depending on identities, belonging to the same company, having the same kind of preferences, etc. Related to this is the case where meetings can be set up spontaneously, based on who is present or what time of the day it is. Resources such as documents or white boards could be shared.

Another type of context-awareness would involve the ability to obtain regional information from the Ad-hoc network. However, this generally requires a gateway or a static device providing this kind of information.

2.5.4 Challenges

Challenges for context-awareness in Ad-hoc networks do not differ from other areas using context information. It can be quite challenging to capture, represent, and process contextual data. This can be even more challenging in an Ad-hoc environment if no

infrastructure for doing this is available, or it is created "ad hoc", or it changes frequently when the network changes. In this case the devices themselves must be able support all necessary mechanisms or they need a standardised context-service in the Ad-hoc network.

Currently the usage of context information is still quite limited since it is very challenging and complex to capture, represent, and process contextual data. The most used types of context information are location, identity and time information. Most of this can be simplified by the usage of the policy based management paradigm for managing context.

2.6 Context and Policy Based Management

Starting from the definition of context (section 2.2), this section gave a current state-of-the-art of on context-aware computing (sections 2.3, 2.4) before moving into context awareness in ad-hoc networks (section 2.5). As discussed in the previous sections, context refers to the physical and social situation in which computational devices are embedded. One goal of context-aware computing is to acquire and utilise information about the context of a device to provide services that are appropriate to the particular people, place, time, events, etc. However, this is more than simply a question of gathering more and more contextual information about complex situations. The essence of context-awareness and context-aware computing lies in a feasible and compatible context model and a powerful framework/architecture to enable the development and deployment of a wide range of context-aware services. The author in chapter 4, proposes a Context Aware Policy Definition Language combined with a Policy Based Management architecture, in order to achieve this. Through the combined use of Policy Based Management and context information, context services can be provided to users in an easier way and more automated way. The context information used in chapter 4, mainly falls in the network context information but the model developed can be easily used for other types of context information.

Before describing the main part of the work in chapter 4, two Policy Based Management systems developed by the author will be described in chapter 3. The first one was

developed for the WINMAN system and did not make use of any context information[11], whereas the second one was to support the CONTEXT system, and had to incorporate context information in it. The two systems developed are related to each other[12] and provided the author with hands on experience in Policy Based Management, which was later used in the development of the work described in this thesis (chapter 4).

.

[11] It was used for network resources management

[12] The design and implementation skills gained from the WINMAN system were later applied to the development of the CONTEXT system.

3 WINMAN and CONTEXT Projects

3.1 Introduction

This chapter describes the policy based management systems, designed and implemented by the author, during two European Information Technology projects. Both provided the author with the required experience and know how in designing and implementing Policy Based Management systems. The first one described, is the WINMAN policy based management system, which was used to control end to end IP over WDM connectivity services derived from SLAs. This introduced the author to the Policy Management for network resource management. The second one described, is the CONTEXT policy based management system, which was used to manage context aware services. This allowed the author to build on the WINMAN Policy Based Management experience and extend it into managing context aware services.

3.2 EU IST WINMAN Project

3.2.1 Introduction

In recent years it has become apparent that the transportation of IP based applications will be a dominant factor in future networks. At the same time equipment for Wavelength Division Multiplexing (WDM) has matured sufficiently to give the very high capacity networks (terabit transport networks [4], [5], [6]) that will be needed for the ever-increasing amount of information. It was soon realised by the telecom industry and the research community that the convergence of those two technologies could form the solution to future networking, offering a universal, reliable and ultra-fast solution. Yet today's transport networks are primarily based on ATM and SDH technologies. Some operators started deploying WDM technology for bandwidth capacity extension between network nodes by means of point-to-point connections. Next generation network architectures focus on eliminating the intermediate layers between IP and optical, thus minimising encapsulation overheads and complexity. The next challenge in order to

deploy these networks is that there is a need to manage their resources efficiently and how to migrate the existing network and management infrastructure to the new one. Management of such networks in an integrated fashion is a large research area [103], [104]. Different approaches to network management have been developed through the years; such include approaches as the TMN, ODP, TINA-C, WBEM and Policy Based Management (section 1.2.3). From these, the Policy Based Network Management was of interest, because of the benefits it offers [137]. Policies are considered to bring routine to everyday network management operations. There is evident need for rules to automate and govern processes and functions. [13] Policies are also needed to define strictly regulated access to various resources. Non-standardised policies for management functions can already exist in a system but they are not created conveniently [14], [15]. Most trends in IP-WDM integration are extensions of the distributed Internet network control approach to the Optical Layer using signalling mechanisms either in an Overlay model or a Peer model. WINMAN proposes an alternative approach for managing Internet services over the Optical Transport Network by extending the telecom-style policy based network management approach to the IP layer over WDM [16].

3.2.2 WINMAN Overview

WINMAN [1] was a European research and development project, which ran from July 2000 to April 2003. It aimed to offer an integrated network management solution for the provisioning and maintenance of IP over WDM end-to-end transport services derived from Service Level Agreements (SLAs). Management requirements for IP over WDM were identified and a two-tier architecture (Figure 5) was conceived for the system that was intended for proof of concept of the WINMAN solution. The first layer contains the technology dependent managers, that is one for the IP technology and another manager for the WDM. These two management systems interact with existing network element management systems through the southbound WINMAN interface. On top of this layer there is a second interface whose purpose is the integration of the above mentioned management systems. This system was in turns integrated in different experimental test beds and fully tested in real life conditions.

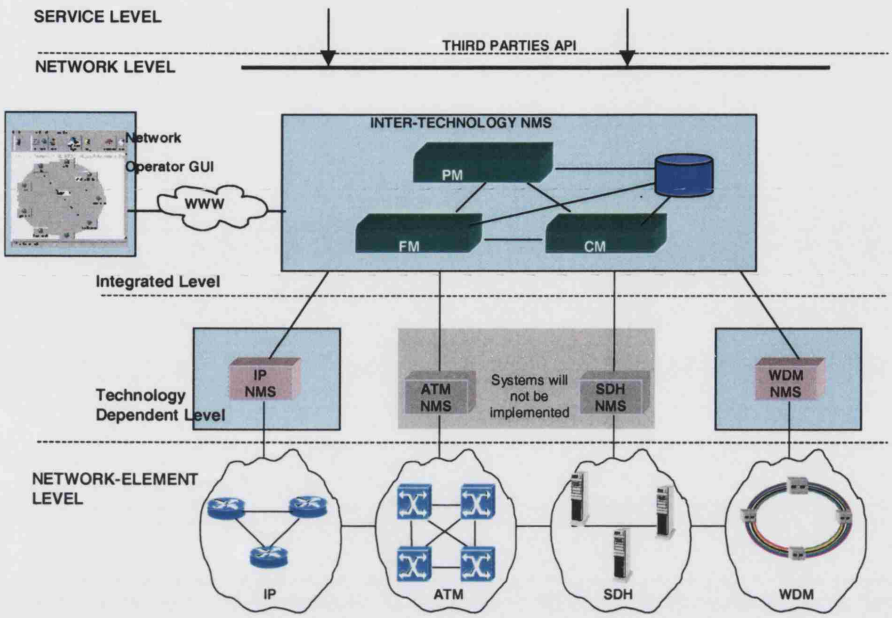


Figure 5: Overview of systems specified and implemented by WINMAN

The WINMAN project's main objective was to provide IP connectivity to third party Service Management Systems. WINMAN implements a management approach for the provisioning of IP based services over optical (WDM) networks with assured quality of service. WINMAN made use of the MultiProtocol Label Switching (MPLS) protocol to infuse a circuit switching behaviour on connectionless IP networks, making them predictable and controllable enough to transport real time traffic. At the same time, the management approach followed by the WINMAN system, allowed its user to create and maintain services without the use of recent signalling protocols that are not mature yet, without excluding their usage in the future.

WINMAN captured the requirements, defined and specified an open, distributed, and scalable management architecture for IP connectivity services on hybrid transport networks (ATM, SDH and WDM) [17], [66]. WINMAN focuses on IP directly over WDM networks[13]. From an implementation point of view, the project addressed the separate management of IP and WDM networks. Per technology domain the integration of the management at Network Management level was carried out. An Inter-technology domain Network Management System (INMS) was implemented as a sub-layer of the

[13] Nevertheless, some evolution scenarios were taken into account, such as IP over SDH over WDM, IP over ATM over WDM and IP over ATM over SDH over WDM (Figure 5).

Network Management Layer to support IP-connectivity spanning different WDM sub-networks and integrate the management of IP and WDM transport networks. The INMS, the IP and the WDM Network Management Systems implement Configuration, Fault and Performance (CFP) management application functions.

3.2.3 WINMAN System Architecture

The business processes addressed by WINMAN, as defined in the TMForum Telecom Operations Map [18], are:

- Network provisioning, which deals with service requests and configures end-to-end connections and services across different technology domains
- Network Inventory Management, whose main functionality is to discover the network resources and topology and maintain a network inventory with all the network resources
- Network Maintenance and Restoration, which maintains a fault topology database, which contains the alarm status of the network, quick fault correlation and localisation, inventory of user defined correlation rules.
- Performance Management, which sets threshold crossing alerts on the available capacity of NEs and links, sets threshold crossing alerts on SLAs, creates reports about network and services status.

The WINMAN solution is performed by several technology specific Network Management Systems (NMSs) and one technology independent NMS. Each one of these technology specific NMSs has a transparent view of the managed network under its responsibility and performs network provisioning, fault management and performance management inside its technological domain. In addition, above all the technology specific NMSs there is an umbrella INMS (Integrated Network Management System) that makes the inter-technology integration. This INMS views all the inter-technology (inter-NMS) connections and coordinates the network provisioning, fault management and performance management between all technological domains. All these three systems share similar functionalities at their network management layer, so the approach adopted by the WINMAN project was to build a Generic Network Management System (GNMS)

[19] with all the common functionalities of the three systems. The GNMS subsystems are shown in Figure 6. From that generic NMS architecture three NMSs have been specialised and further refined, and developed by the WINMAN consortium.

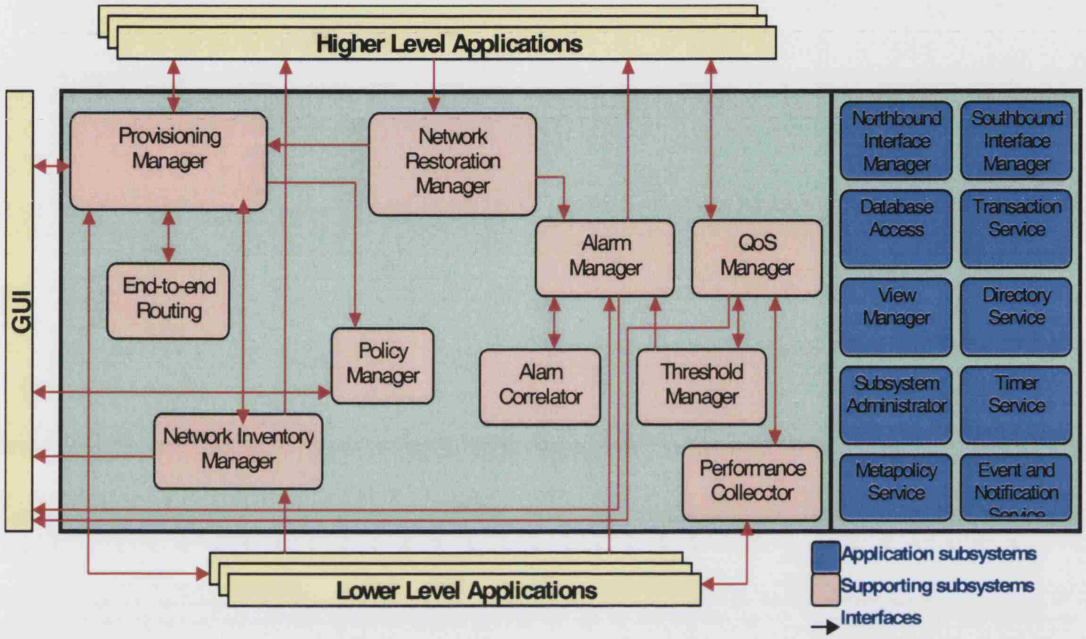


Figure 6: WINMAN Generic NMS Architecture

The Provisioning Manager is in charge of provisioning the IP services. It manages the provisioning process, including scheduling. The End-to-end Routing performs the design of the end-to-end connections inside its own network, taking into account QoS constraints and routing policies.

The Network Inventory Manager's responsibilities are to store, update, maintain and provide information about the data that WINMAN uses related to network physical resources, according to the information received from the network element layer and the GUI system. The other blocks will use these data.

The Policy Manager [59], [60], [61], [62], [63], [64], [65], [67] is responsible for managing and providing policies, necessary to make decisions in a variety of actions. For instance, it checks a provision request against the correspondent policies. Alarm and performance mechanisms can be policy oriented. The routing and the restoration mechanisms can also be controlled by policies.

The Alarm Manager receives alarms from the Lower Level Application, triggers alarm correlation, stores alarm data and distributes alarms to other systems and subsystems. The Alarm Correlator filters, correlates and evaluates the alarms to find out their root cause and generate new alarms, sending the results to the alarm manager

The QoS Manager monitors and analyses the QoS data of the paths provisioned in the network and sent by the Lower Level Applications. It also provides performance data to the GUI and the Higher Level Applications. The Threshold Manager checks counters against the defined thresholds in order to generate alarms and reports if the thresholds are passed.

The Performance Collector collects performance data from the Lower Level Applications. The Network Restoration Manager is responsible for the network restoration actions taken in order to prevent the disruption of the provided connectivity services. These actions are taken when alarms from the Alarm Manager arrive to the Network Restoration Manager subsystem.

The WINMAN system offers a northbound interface where Service Management Systems (SMS) might plug-in and request IP connectivity services. This interface complies with the Connection and Service Management Information Model CaSMIM [20] standard from the TMForum. The system also interfaces through its southbound interface with one or more Element Management Systems (EMSs), through an interface compliant with the Multi-Technology Network Management (MTNM) standard [21].

3.2.4 The Policy Manager Component

The Policy Manager[14]was designed (and implemented) by the author to interact with the other WINMAN system components through two types of interface classes. These components were designed by other members of the WINMAN consortium but since they had to interact with the Policy Manager, the Policy Manager component functionality was influenced by this. One interface type is a "server interface class" which is unique and is used by the other components that need the Policy Manager for their operation.

[14] The Policy Manager Component has the same role as the Policy Decision Point in the IETF architecture (section 1.3).

This interface could be used by the End-to-End Routing (E2Erouting) Component (and the other interacting components) to request if a route can be used (or place authorisation requests in general). The other types of interface classes are the "client interface classes". These classes are as many as the components making use of the Policy Manager functionality. This was done in order to provide dedicated interfaces to all the components using the Policy Manager and hence allow for a smoother integration with the components developed by other partners. The external interfaces related to the Policy Manager are depicted in Figure 7.

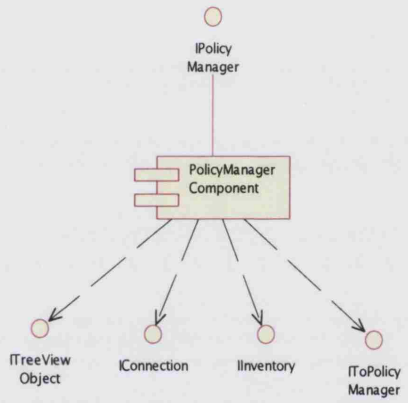


Figure 7: The Policy Component

From the use cases developed during the design phase [19], the Add Policies, Modify Policies and Release Policies use cases, were supported by the interface between the Policy Manager and the WINMAN Operator (WO) (at the policy console[15] which is policy editor provided by PONDER [22], [23] modified to suit the WINMAN needs). This interface is also provided by PONDER (IPolicyManager[16]). The Provide Policy Report use case is satisfied by an internal component. On the other hand, the support to the use case Execute Policies is provided by the interface between the Policy Manager and the Provisioning Manager and the Policy Manager and the E2E Routing. This functionality is offered by the IToPolicyManager[17] Interface. The support to the use cases

[15] The PONDER editor is the Policy Management Tool of the WINMAN Policy Based Management System. This is used to author the policies and generate the policies code to Java. Once that is done, the policies are received by the Policy Manager Component.

[16] IPolicyManager is an abbreviation for Interface PolicyManager

[17] IToPolicyManager is an abbreviation for Interface To PolicyManager

Find Policy and Check Policy is provided by internal methods of the class Policy Manager, because a local policy repository is assumed. Finally the use case Activate Policies can be implicitly supported by the same methods supporting the Execute Policies.

3.2.5 The Policy Component

As a general remark, the Policy Manager's operation was foreseen as a support component that becomes active in response to internal[18] or external events. These second type of events are to be raised by the Provisioning Manager and the End To End Routing module. The Performance Manager and the Scheduler generate events for the Policy Manager.

Policies allow the WO to modify the system behaviour dynamically; they are persistent but can be changed on the fly; this means than the WINMAN system's behaviour can be changed without recompiling, just by adding/changing policies, making the system behaviour flexible. The insertion or deletion of a component is known to the rest of the components by means of the infrastructure naming service. As a supporting component, Policy Manager adds value to the system by providing services to the rest, but its presence is not a prerequisite for system operation.

The idea is that Policy Manager's clients (WINMAN components that use Policy Manager functionality) would have a default "permit any" policy that allows their operation in case of Policy Manager absence. If the Policy Manager is running, then the policy-based value is added to the system. The Policy Manager interacts with the End to End Routing and the Provisioning Manager acting as a server: the interface with these components is carried out by the class IpolicyManager which contains two methods: getAuthorisation() and get Action().

On the other hand, the Policy Manager also interacts with the Inventory, the EndToEnd Routing and the Provisioning Manager acting as client to them. This is done through the interfaces offered by these components. The ProvisioningManager's IToPolicyManager interface is used in order to retrieve provisioning requests, and the EndToEnd Routing

[18] This could be the WINMAN Operator (WO)

IPolicytoRouting interface to retrieve routes. Finally, the GUI for the Policy Manager is a modified Ponder editor Graphical User Interface (GUI), which has been integrated with the general WINMAN GUI.

3.2.6 Policy System Interfaces with WINMAN

Due to the fact that the Policy Management System in WINMAN, if present, is acting as a supporting component to the other components, its interfaces[19] were determined by the required interactions with those components. The IPolicyManager interface provides two main functions: the getAuthorisation and the getAction. The Provisioning Manager module executes the getAuthorisation() method when it is necessary to retrieve the policies associated with a given provisioning request. This method returns a byte indicating if the request is accepted or not, with an associated error code. This error code informs the reason why the provisioning request is not accepted, allowing the provisioning manager to take the appropriate corrective action before denying the request to the user. The getAction() method is issued by external actors, for example the WO or the Performance Manager component when some condition previously programmed is met.

When the Policy Manager executes this method, it tries to verify whether the triggering conditions are met. An example of such conditions are time, existence of special traffic events (eg. a football match TV multicast), other variables reflecting the network status (eg. some congestion threshold), etc. If some logic function of the aforementioned conditions is met (as specified by the relevant policy rule), the Policy Manager issues the method provisioningAction(), that's why an interface to the Provisioning Manager is needed.

The IToPolicyManager interface provides two main functions: the provisioningAction() and the routingAction(). The provisioningAction() is used in order to send the appropriate configuration commands to the Provisioning Manager in order to reconfigure the network. This network reconfiguration may entail the modification, release or establishment of several connections. The Provisioning_object input is related to

[19] These interfaces are well defined and are proprietary to the WINMAN system.

obligation policies. It returns a Byte indicating if the intended configuration could be performed or not, and an error code. This configuration may be fired for example by the following policy:

'**if** Not Business Hours and Business LSPs Usage < 50%
then *Change Configuration to Non Business Hours*
else *Notify Billing for Out of Business Usage*'

This policy permits that a trivial Scheduler policy could be checked against network conditions to take the appropriate actions, that can be different from the initially intended one, in this case change network layout from business hours to non-business hours. The routingAction is used in order to send the appropriate configuration commands to the EndToEndRouting Manager in order to perform some needed routing action.

3.2.7 Policy Manager Entities

The internal structure of the Policy Manager (Figure 8) is composed of the Ponder Editor/Compiler, an LDAP repository and the Enforcement Agents (EAs), which implement the actual PDP/PEP functionality. As mentioned in section 3.2.4, the Ponder editor[20] was used as the Policy Management Tool for WINMAN. In order to do so, an interface on the Policy Manager Component was developed (I_PolicyEditor), through which a notification of a policy modification could be sent to the Policy Manager.

The EAs are pluggable components that add functionality to the system using the common functions provided by the EA Coordinator. EAProvisioning and EARouting are two possible examples of EAs.

[20] The editor was used for authoring the policies and compiling them in Java classes.

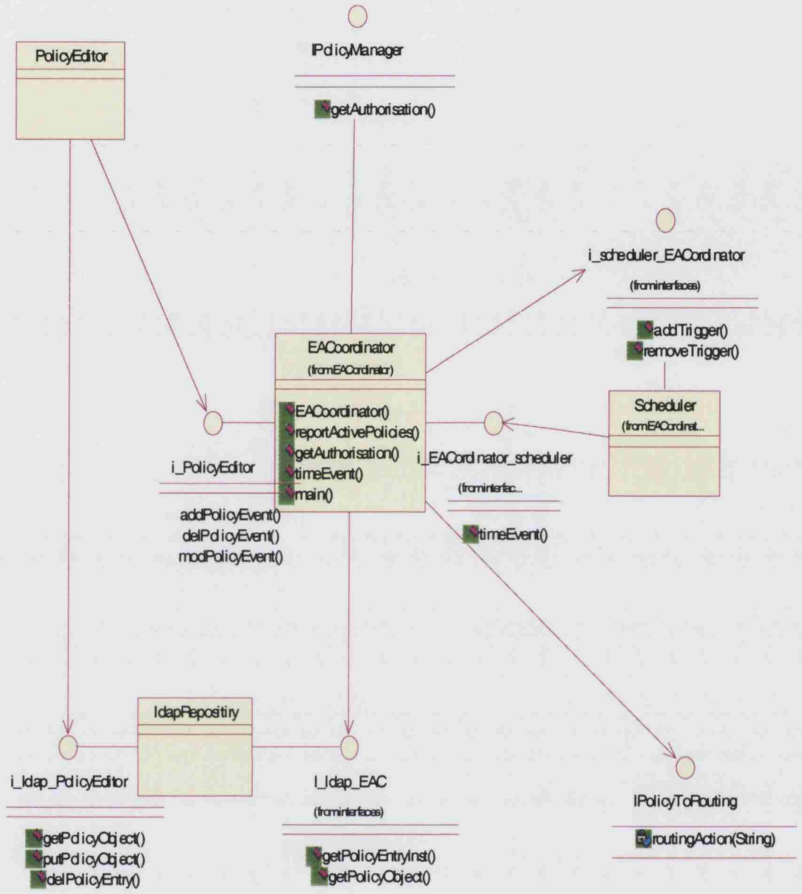


Figure 8: The Policy Manager

The EA Coordinator is the core of the Enforcement Agents functionalities. It provides common functions needed by these EAs. It implements the getAuthorisation() and getAction() methods, selects the appropriate EA to perform a given PDP/PEP function, accesses external components and internal Policy Repository in order to retrieve/store the Policy and Provisioning Objects. It also performs the policy conflict check.The EA Provisioning entity provides the needed PDP/PEP functionality for provisioning. It communicates to Provisioning Manager interface and executes the provisioningAction() method. Its main function is to perform the enforcement and authorisation for provisioning policies. The EA Routing provides the needed PDP/PEP functionality for routing. It communicates to EndToEnd Routing Manager interface and executes the

routingAction() method. Its main function is to perform the enforcement and authorisation for routing policies.

3.3 EU IST CONTEXT Project

EU IST Project CONTEXT [24] is an abbreviation for "active creation, delivery and management of efficient CONTEXT aware services". It was a medium-sized project including eight partners from Europe. It started on September 2002 and lasted for 2.5 years. Its main objective was to specify and design models and solutions for an efficient provisioning of context-based services making use of active networks on top of fixed and mobile infrastructure. CONTEXT promised to design, develop and assess innovative models and solutions for an efficient provisioning of context aware services making use of active network technology on top of fixed and mobile networks.

As described in chapter 2, the environment in which a service application computation takes place is called its context. A context may either refer to aspects of the physical world or to conditions and activities in the virtual world. Context awareness enables a new class of service applications in a relatively new research and development area known under the terms pervasive computing or ambient intelligence. So far proposed service applications can help users navigate unfamiliar territory, find nearby restaurants, receive messages in the most useful and least intrusive manner, find people with similar interests, and the like. Adding context information in these applications reduces the amount of human attention an application needs to service the user's requests managing context data. In addition, little to no attention has been paid to giving users control over the release of their own context information. To date, there has been little work in context-aware service applications and the development of context-aware service applications has been hampered by the need to develop a custom context infrastructure for each application. CONTEXT removes this obstacle by placing the context functionality in the network infrastructure in a form of active components. The CONTEXT project have claimed that: "*Our project is making context services an integral part of the pervasive networking infrastructure in a form of context management servers, active deployment and use customize and personalize network enabled service,*

thus substantially advance the current state of the art. Applications could then interact with context services to obtain the information they desire without worrying about the details of context management. Further, the costs associated with introducing new context sources can be amortized across many service applications.

The project is developing a novel service application execution platform, called Active Context Middleware (CONTEXT middleware), which enables, manages and continually provisions context-aware service applications. It would span over multiple network technologies, which will allow users to dynamically access customized and personalised context aware services from any type of terminal within the network. The CONTEXT project focuses on the development of a uniform service middleware that will fulfill the following requirements:

- *Network infrastructure independence*
- *Span over a number of heterogeneous network technologies (IPv6, WLAN), via specific service adaptation components*
- *Support for context management*
- *Support for context aware services*
- *Rapid and adaptive active service deployment*
- *Fully distributed architecture*
- *Service programmability and extensibility*
- *System of loosely-coupled, dynamically-bound components*"

The approach to develop an adaptive service and context execution environment for next-generation networks, which is inherently distributed and programmable, is based upon using active technologies. The term active technology is used to include the recently developed paradigms of Active Networks (ANs) [88], [89], [90], Programmable Networks (PNs) [91], Mobile Agents (MAs) [92], [93] and Semantic Networks (SNs) [94], [95]. Although these concepts were introduced by different research communities to address different problems, they have started overlapping in focus and applicability [96], as they are being developed further. Next-generation service networks can benefit from

active networking paradigm with the dynamic deployment of network services that can be tailored to the user's requirements using context. For example, ANs distribute code to proxies at the edge of the network, as well as, to the mobile device, thus reducing the number of necessary network transactions to provide a service, by local processing and local service customisation. In this respect, the CONTEXT project leverages existing work on AN-assisted services and applications, and further extends the applicability of AN in next-generation network infrastructures in support for context aware services.

The CONTEXT project main objective was to design, develop and assess innovative models and middleware solutions for an efficient provisioning of context-aware services making use of active systems technology on top of fixed and mobile networks. This allows for the composability and dynamic adaptability of current and future context aware services for the benefit of the global consumer. From the technical point of view the CONTEXT solution spans across three domains, namely: the Service Layer (SL) domain, the Active Applications (AA) domain and the IPv6 domain. In the service layer domain, modelling of the information that expresses the context of services and establishing a framework for the creation of context aware services is being dealt with. The AA based solution, with appropriate APIs to control the IP domain, will allow the actual delivery of policy based context aware services.

In the Service Layer domain, CONTEXT aimed:

- to identify mechanisms for the definition, exchange and acquisition of contextual information
- to identify mechanisms for creating context sensitive services (binding of contextual information to existing services, thus creating context-sensitive services)
- to identify mechanisms for provisioning 'contextualised' services (based on the available provisioning means of the existing service infrastructure of the domain)
- to propose a policy-based framework for service activation, including configuration of the relevant servers according to the personalised user profiles
- to specify and implement the interactions with underlying Network Management Systems by deploying relevant configuration for service delivery

- to allow the enforcement of monitoring policies for Service Level Agreement management

In the Active Application Layer domain the CONTEXT project aimed:

- to enhance active network technology in order to provide efficient delivery of context based services especially in the mobile network environment. This will be accomplished by specifying and developing an API on top of existing AN nodes to allow the following functionality:
- to enable a Policy-based active node management
- to get information from the data plane (packet) and also from the management plane as needed by the service

In the IPv6 domain, CONTEXT project aimed

- to identify the functionality needed at the IP domain in order to allow the efficient delivery of context based services
- to specify and develop an API exhibiting a subset of the functionality of the IETF FORCES /P1520 L reference model to allow the programming of the behaviour of the IPv6 network elements

In addition the CONTEXT project had the following objectives:

- Provide open low-level access to network elements (routers, switches, base stations) by defining appropriate interfaces to enable mapping of applications' requests onto the behaviour of underlying Ipv6 network
- Test and validate the efficiency of the developed service execution platform for the provisioning of context aware services, following a number of different trial scenarios (i.e. the super woman, crisis help and business creation context aware service scenarios)
- Disseminate the project results within the IST community and relevant industrial forums, as well as, contribute to the activities of various standardization bodies

(IETF, OMG, IEEE) and research forums like WWRF- (Wireless World Research Forum) and 3GPP (3rd Generation Partnership Project) and in particular Open Service Architecture Working Group (TSG CN WG5).

To cope with the aforementioned project objectives, an architectural model was adopted, relying on open programming interfaces for network devices, such as switches and IP routers. A programming interface specifies an abstraction of network resources, in other words being, a software representation of these resources. This is the lower level interface and it mainly incorporates virtual network devices. On top of this, a set of generic network services controlling the functionality of the network is foreseen, which export their functionality through an upper-level interface. Finally, on top of the upper interface, a set of value-added services may be realised tailored to the user's needs. A similar architecture is currently being developed in the IETF FORCES (Forwarding and Control Elements Separation) working group [25], and it is planned to adopt that terminology.

3.3.1 Policy Based Service Management

The Policy Manager was designed (and implemented) by the author to interact with the other CONTEXT system components, which were designed by other members of the CONTEXT consortium. Because of this, the Policy Manager functionality and design was influenced by requirements set by other components and the IETF Policy Management architecture (section 1.3). The solution for the Policy-Based Service Management of the IST-Context Project relies on a robust and flexible architecture accommodating the service management systems and like correspondence new types of services. According to the CONTEXT project: *"The CONTEXT system architecture should exhibit the following logical attributes:*

- *Open:*
  It is built on standard interfaces between architectural components.
- *Flexible:*
  Supports the incorporation of any new context services that complies with the specified interfaces, and allows the modification of the system behavior by means of user defined policies.

- *Modular:*

  Its architecture is based on components, standardized and oriented to context services.

- *Scalable:*

  Achieved with the separation of different building blocks of technology specific and technology independent functionality, and through of the minimization of data redundancies.

- *Distributed:*

  The selection of a component based architecture defined to run on top of a standards-based object request broker guarantees the transparency of the system to the location of components."

The architecture for the Service Management layer can be seen in Figure 9.

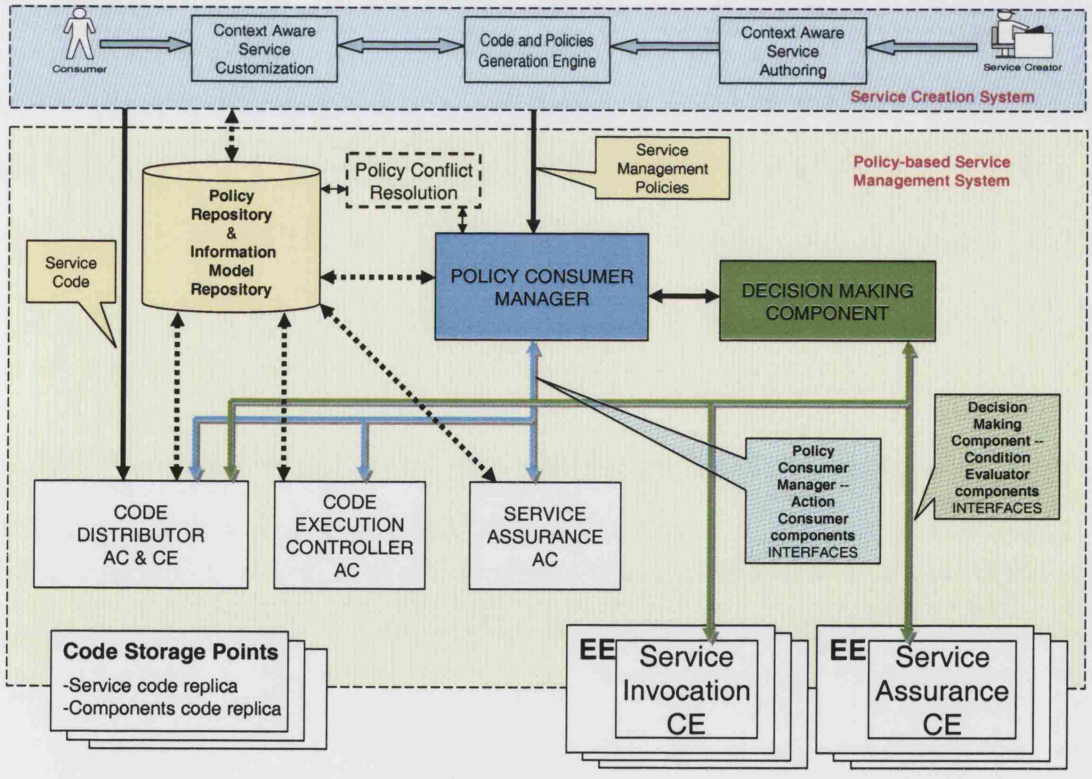


Figure 9: IST-CONTEXT Policy-Based Service Management Layer Architecture

The following sections describe the Context Policy Based Service Management[21] [58] (PBSM) components appearing in Figure 9, from which the author designed and implemented the core of the Policy Based Management system; the Policy Consumer Manager, the Decision Making Component and the Policy Repository & Information Model Repository. When a new service is created, the Service Creation System sends the Service Management Policies to the Policy Manager. The Service Code generated by the Service Creation System is delivered to the Code Distributor Components. The Management Policies generated by the Service Creation System for the new service are used to drive the distribution, invocation, execution and assurance of the service.

3.3.2 Policy Consumer Manager

The Policy Consumer Manager (PM) is always waiting for new policies to arrive from the Service Creation Phase. These policies are related to the Service Management. The specific functionalities required by these policies will be explained later on this document as well as their structure.

Once a new policy, or group of policies, arrives as an XML document, the PM parses it to Java and sends it to the Policy Conflict Resolution Component[22] in order to check for conflicts with other policies that are already in the system.

Once all the checks have been completed, PM process the different groups of policies in order to check which of them have to be processed immediately, or which of them will be stored for further processing (policies will specify if they have to be sent to be evaluated following some sequence or concurrently, etc). PM sends the policy conditions to be processed to the Decision Making Component in order to start monitoring the parameters set by the condition part of the policies.

When the Decision Making component decides that conditions of a policy are met, it notifies the PM so that the latter can retrieve the actions associated with that condition from the policy repository, and communicates them to the appropriate Action Consumers

[21] The CONTEXT project Policy Based Service Management system is described in more detail in the appendix A.2

[22] Not clearly depicted in Figure 9 because was not implemented. (Mentioned for completeness purposes)

in order to enforce them. Before sending a policy action, the PM is going to check if the corresponding Action Consumer responsible to enforce it is installed or not in the target. In case that it isn't, it is also its responsibility to order the download of this Action Consumer from a management code repository[23] and install it before sending the action. The PM can search in the Information Model objects what Action Consumer is installed or available in the management code repository.

However, it might also be possible that some policies don't have conditions to monitor; in this case the policies are going to be enforced when received.

3.3.3 The Policy Repository Component

The Policy Repository is the logical place where policies are stored and fetched when needed. In this repository all policies related to the CAS Management can be found. They will be stored in two formats: Java Code and XML. The main reason for this decision is that storing the policy in XML will help in its creation and modification, whilst storing it in Java Code will make the execution of the actions faster, since there would be no need of recompiling the policies before being sent to the corresponding Action Consumer.

3.3.4 The Policy Conflict Resolution Component

The Policy Conflict Resolution component (PCR) has the responsibility of maintaining the consistency of all policies introduced in the system. One example of inconsistency would be that two policies require opposite actions when the same conditions are met. The functionality of the PCR is to detect these situations and only accept new policies when they don't introduce inconsistencies to the system.

It has to be said, that the algorithms to detect and solve inconsistencies are not straightforward. For this reason the implementation of this component was not committed and the component has been mentioned only for completeness.

[23] Does not belong to the Policy Based Management System and hence is not depicted in Figure 9.

3.3.5 The Decision Making Component

This Decision Making Component (DMC) has two main functionalities:

- The DMC is intended to receive the policy conditions from the PM. The conditions contain information about which components (Condition Evaluators) are responsible for monitoring and evaluating each variable and requirement that composes the policy condition. These Condition Evaluators[24] (CE) can be installed in active nodes or not. In case that these CE are not yet installed, it is also the task of the DMC to request their downloading from a management code repository in a dynamic way. The DMC, having checked that the CE is installed and available, will send to the CE a message containing the events or variables, and the possible constraints that its values must accomplish to consider the condition met. This message will also include an identifier of the policy to which this condition belongs. The CE, having received this information, will start the monitoring and evaluation process.

- The DMC is also intended to receive the monitored values from different CEs in order to aggregate and process them to determine whether a given policy condition is met, or simply to receive the result of conditions evaluated at the CE itself. Anyway, the DMC is the entity entrusted to notify the PM that a policy condition is fulfilled. The messages received from the CE will also include the identifier of the policy to which the condition evaluated belongs (this is the same identifier received by the CE within the message coming from the DMC that started the evaluation process). This identifier will be used to find the action associated to the condition evaluated inside the policy repository.

3.3.6 The Action Consumers & Condition Evaluators

The Action Consumers and Condition Evaluators were not implemented by the author but are presented in this section for completeness, since they contain the Policy Enforcement

[24] A CE could be used to monitor users joining a dynamic network. This can be done by installing a CE on the DHCP server of the network, which would be monitoring for DHCP requests coming from a MAC address belonging to a specific user. In this way, a possible policy applicable to that user could be triggered.

functionality for the CONTEXT Policy Based Service Management System. Regarding the Service Management Layer, there are four main functional blocks identified. These are:

- Code Distribution and maintenance
- Code Execution
- Service Invocation
- Service Assurance

The blocks are detailed in the Context Project Public Deliverable D3.2 [97] (D3.2). These functional blocks are policy driven. This implies, that its functionality is influenced in a policy ruled way. So the work developed was to design a policy management system that handles, manages and applies the service management policies that would rule the behaviour of the system, and particularly the efficient delivery of the functionalities identified.

The components that are technology and policy specific are the Action Consumers and the Condition Evaluators. These components are the ones in charge of understanding the policy semantics, evaluate the conditions (CE) and apply the corresponding actions (AC). So, in order to be able to provide the different functionalities described in D3.2, specialised Action Consumers and Condition Evaluators were proposed to be developed and used.

Also, the policies delivered by the Service Creation System will be heavily linked to these functionalities. So the AC and CE are grouped taking into account the functional components they are related to.

The proposed AC and CE for each functionality are:

- Code Distributor ⟹ *Code Distribution Action Consumer* and *Code Distributor Condition Evaluator.*
- Code Execution Controller ⟹ *Code Execution Controller Action Consumer*
- Invocation Service Listener ⟹ *Service Invocation Condition Evaluator*

- Service Assurance $\Longrightarrow$ *Service Assurance Condition Evaluator* and *Service Assurance Action Consumer.*

It should be taken into account that the Condition Evaluators could not be a unique component but a set of different components specialised for particular tasks or different instances of these components installed in different points of the network. For example, service assurance could require different Condition Evaluators each one of them specialised in monitoring different performance or quality parameters, or installed in different nodes. Another example could be service invocation where different Condition Evaluators are required and each one of them is specialised in monitoring or listening to different kinds of invocation signals or events and they are installed at different points.

3.3.7 Policy Syntax

In CONTEXT, the policies are expressed and transmitted in extended markup language (XML), XML (eXtensible Markup Language). XML has recently emerged as a widely accepted way of representing and exchanging structured information, The principal technical strengths of XML are that it has a text-based representation, which imposes few restrictions on network technology or protocols and that, through the use of XML Schemas[25], it has sufficiently strict syntax to permit automated validation and processing information in an unambiguous way.

The policy structure used in the Policy Based Service Management (PBSM) is based on the IETF Policy Core Information Model but simplified for policy processing. Hence, the size of policies is considerably reduced and their processing is fast and simpler than the Policy Core Information Model (PCIM) [45].

XML allows the definition of new markup tags as well as constraints on their relationships through the use of templates. XML Schema allows more control over the way XML documents are specified. Datatypes are supported and there is the ability to specify relationships and constraints between different elements of a document.

[25] See reference [98] and section A.3 in the appendix

The structure of XML documents is dictated by the XML Schema against which such documents are validated. Hence, hereafter we will describe the main aspects of the XML Schema defined for CONTEXT Policies. It just describes those functional domain independent elements.

The Schemas, the concrete syntax and basic structure are extensively described in this section. In CONTEXT all XML policies are related to a customized context –aware service and contains the following fields:

- Two identifier elements for service type. These include the Service identifier and the subscription identifier. These two elements are used in order to identify a unique service.
- Three identifier elements for policy type. These include the Policy Identifier, Atomicity Identifier of a policy and the Policy Sequence Position identifier. These elements are used in order to identify a policy.
- An identifier element for policy group. Each of the policies can be categorised in Groups. This element is used to identify the group to which a policy belongs to.
- Four policy management information elements. These elements are used in order to denote the time for which the policy will be valid.
- Three additional elements more related with policy conditions and their parameters. These elements are used to identify the conditions and the threshold values by which the policy will be triggered.
- Five additional elements more related with policy actions and their parameters. These elements are used to identify the actions the policy will be perform when triggered.

Table 1, shows the CONTEXT XML-Policy Information Structure description and Figure 10 shows the shows the XML Schema for the policy.

Policy Model Structure

- Policy_Type
  - Service_Id
  - User_Id
  - Policy_Id
  - Is_Atomic
  - Policy_Sequence_Position
  - Policy_Group
    - Group_Id
    - Number_of_Policies
    - Is_Atomic
    - Policy_Group_Sequence_Position
  - Validity_Period
    - Time_Period
    - Month_Of_Year
    - Week_Of_Month
    - Day_Of_Week
    - Hour_Of_Day
  - Condition
    - Condition_Object
      - Event
        - Name
        - Event_Variable
          - Name
          - Type
        - Monitoring_Component
          - Component_Name
          - Component_Location
        - Monitoring_Parameter
          - Name
          - Type
          - Value
      - Simple_Variable
        - Name
        - Type
        - Monitoring_Component
          - Component_Name
          - Component_Location
        - Monitoring_Parameter
          - Name
          - Type
          - Value
      - Aggregated_Variable
        - Name
        - Type
        - Operation
    - Condition_Requirement
      - Type
      - Condition_Object
        - Event
          - Name

- Event_Variable_Name
- Simple_Variable_Name
- Aggregated_Variable_Name

- Requeriment_Parameter
  - Name
  - Type
  - Value
- Evaluator Component
  - Component_Name
  - Component_Location

- Evaluation_Parameters
  - Evaluation_Method
  - Evaluation_Scheduling
  - Start_Time
  - Evaluation_Periodicity
  - Time_Out
  - End_Time

- Global_Enforcement_Parameters
  - Enforcement_Scheduling
  - Global_TimeOut

- Action
  - Name
  - Action_Parameter_List
    - Parameter
      - Name
      - Type
      - Value
  - Enforcer_Component
    - Component_Name
    - Component_Location
  - Enforcement_Parameters
    - Start_Time
    - End_Time
    - Enforcement_TimeOut
    - Enforcement_Priority
  - Success_Output_Parameters
    - Action_Result
      - Value
    - Output_Parameter
      - Name
      - Type
      - Value

Table 1: General Policy Information Model Structure

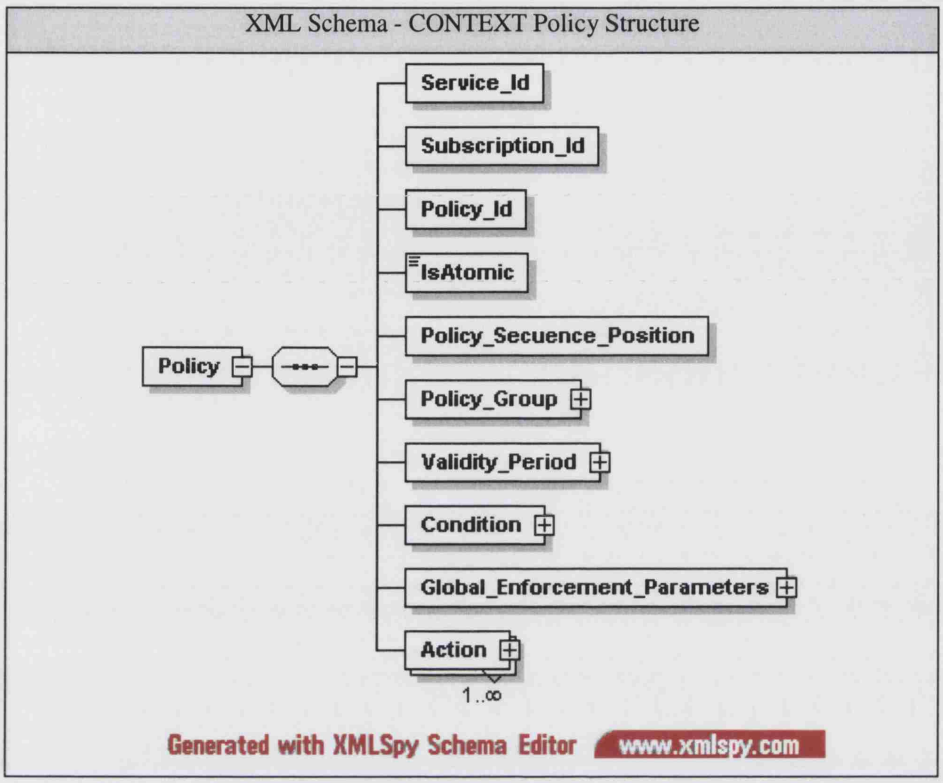


Figure 10: CONTEXT-General Policy Structure

3.4 Conclusions and Summary

This chapter described the policy based management systems, designed and implemented by the author, during two European Information Technology projects. The first one described is the WINMAN policy based management system, which was used to control end to end IP over WDM connectivity services derived from SLAs. This system's main aim was to manage network resources by acting as a supporting component to the WINMAN developed network management platform. The second system described, was the CONTEXT policy based management system, which was used to manage context aware services.

Both of the designs of these systems posed different challenges to the author but were architecturally related[26] and provided hands on experience in the Policy Based

[26] Both of the WINMAN and CONTEXT project Policy Based Management Systems were designed and implemented following the IETF Policy Based Management Architecture, which is the standard followed by most of the commercial and custom-built Policy Based Management Systems.

Management field. Both of these designs have influenced the design and implementation of the work carried out by the author in chapter 4.

As mentioned in section 3.2.5, the Ponder language was used for expressing policies for the WINMAN system. In order to do so, Ponder was extended to support Network Management policies[27]. During this work, it was realised that Ponder at the time, was not mature enough and was lacking supporting tools[28] and documentation making its use difficult. Furthermore, the Ponder Policy Schema provided with the Ponder Toolkit was designed to only support a very old version of the Netscape Directory Server, and hence making the language less portable to other systems.

The WINMAN experiences were later on found to be useful during the design and development of the CONTEXT Policy Based Management System. In order to avoid the complications posed by the use of the Ponder in WINMAN, the use of XML was preferred instead. By using XML for expressing policies, several benefits can be gained as described in section A.3. The use of XML for expressing policies was also adopted by the author in the work described in chapter 4.

[27] Ponder was developed in order to specify management and security policies for distributed systems

[28] The Ponder framework was composed by just the policy editor

4 Novel Framework for Management of Context-aware Services and Networks

4.1 Introduction

The previous chapters mainly presented the current state-of-the-art regarding policy-based management and context awareness. Apart from their relevance to the topic of the thesis, some interesting challenges were identified and are proposed to be solved to some extent in this thesis. In summary, the literature survey (chapters 1 and 2) showed that:

- Not much research in Context-aware Computing concentrates on the network context (2.4). To compensate this, a novel network-centric context-aware management approach is proposed by the author and described in the following sections.
- No practical context model is currently available in the context-aware computing field. Previous work is mostly concerned with system architectures for human computer interaction (section 2.2). A novel object oriented policy-based context modelling is proposed, which is in line with the underlying network management approach, and therefore helping to achieve consistency during context-aware service provisioning.
- No policy definition language (PDL) has yet been developed to follow the IETF policy definition (sections 1.4 and 1.5). A policy definition language following that is proposed in this thesis. This language is generic, i.e. it can be used to define both service-level rules as well as network-level rules.

This chapter describes the first version of the context-aware system developed by the author in order to address the aforementioned issues

It contains several novel attributes, such as:

- Synthesis of a novel categorisation of context awareness

  As discussed in section 2.4, the author has synthesised a novel categorisation of context awareness based on the granularity of context information, which is

composed by two categories, an application-related and network-related category. This thesis is focusing in the network aware category and more specifically the subcategory known as auto configuration, which is of great interest.

- Implemention of an Object oriented Context Information Model

  Based on the new context awareness categorisation and Dey's definition (section 2.2) of context, a novel object oriented context information model has been designed and implemented. This information model is influenced by the IETF PCIM. Therefore, the modelling of context information starts from ContextVariable, which inherits the abstract class available in the PCIM class PolicyVariable.

- Design and implementation of an IETF compliant Policy Definition Language capable of expressing context aware policies

  Based on the Context Information Model developed by the author, a novel Policy Definition Language (CAPL) has been designed and implemented. This language is designed based on the IETF PCIM (Policy Core Information Model) and its extension, directions which are followed by most of the work carried out in the policy-based management field. The major objective of such information models is to bridge the gap between the human policy administrator who enters the policies and the actual enforcement commands executed at the network elements.

- Creation of a Generic Policy Based Management Architecture

  A novel Generic Policy Based Management Architecture capable of understanding and interpreting the policies expressed in CAPL has been designed and implemented by the author. This allows the authoring and application of context aware policies on a network.

Other novel components of the author's work include the monitoring of the policy conditions and the distribution of the policy actions, to the policy enforcement points through the use of an Active Networking technology[29].

[29] This will be discussed in chapter 7, where the system will be tested through the use of two scenarios/services

4.2 A Generic Policy Based Management Architecture

In order to accommodate the aforementioned requirements, a generic Policy Based Management Architecture (PBMA) has been devised. This architecture (Figure 11) is closely related to the IETF model but specialised to take context information and services into account.

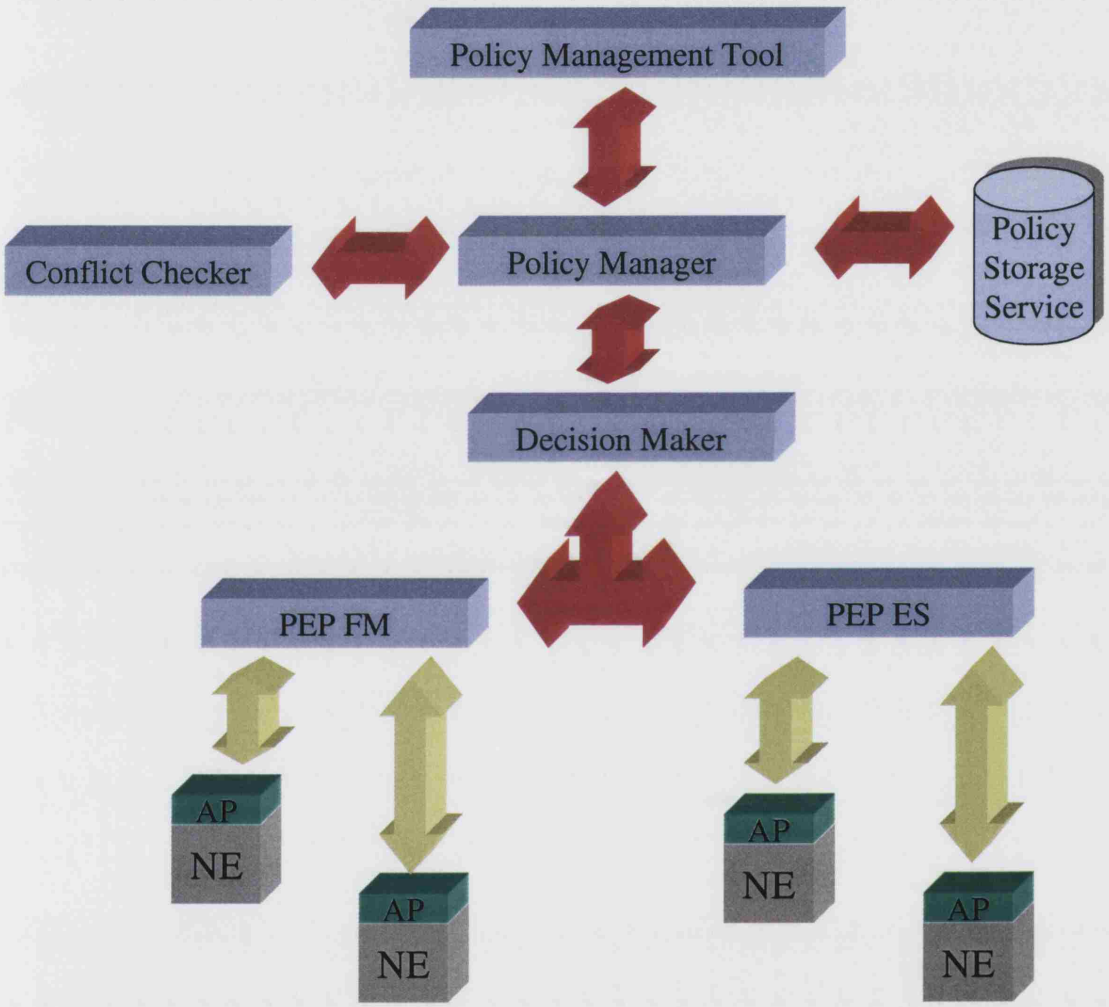


Figure 11: A Generic Policy Based Management Architecture

It comprises of a Policy Management Tool for authoring the policies, a Policy Manager component that manages the policies, a Decision Making component for making decisions, a Policy Storage Service for storing the policies, a Conflict Checker component for identifying policy conflicts and some policy enforcement points (PEP)

specialised to the service to be supported. Finally, at the bottom of the figure, the network elements (NE) are depicted. These are active routers (running an active platform (AP) on top of them). The active platform will allow for the monitoring of the network and also the enforcement of the policies themselves. The detailed description of each of the components will be carried out in the following sections.

4.3 Requirements for Context Modelling

Apart from a clear context definition as already given in Chapter 2, context also needs to be classified in such a way as to assist the design and implementation of the context classes when building the context information model.

4.3.1 Classification of Context

In this thesis, the proposal similar to the one proposed by Dey is followed, i.e. the author uses *location*, *identity*, *time*, and *activity* as *basic* context types for characterising the situation of a particular context entity, as depicted in Figure 12.

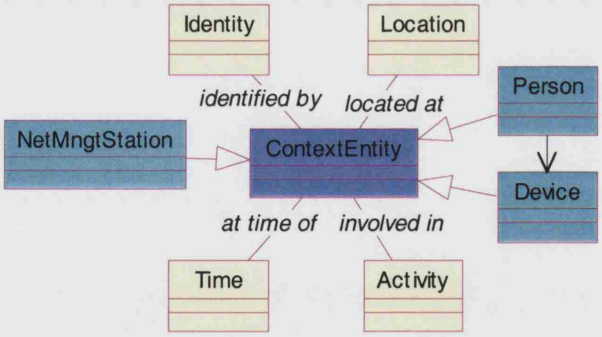


Figure 12: Classification of context in entity terms

This context classification (Figure 12) clearly answers the questions of who (*Identity*), where (*Location*), when (*Time*), and what (*Activity*) for a specific context entity (*ContextEntity*). This contributes to the horizontal classification of context information. The vertical classification of context information is about a context entity itself, which

can be categorised, as typical instances, as: *persons* including end users with different roles, service provider administrator, network administrator, or *devices* covering either mobile handsets, fixed IP routers, wireless LAN access point, etc or virtual entities like *network management station.* Furthermore, an object-oriented design of these context types is also carried out in this thesis, which was not carried out by Dey.

An important feature of this, is that some of the context types can also act as indices to other sources of context information. For example, given a person's identity, one can acquire many pieces of related information such as phone numbers, addresses, email addresses, list of friends, relationships to other people in the environment, or even the emotion of this person. This kind of information, which is related to the basic context, is called *derived* context (e.g., the email address) of that entity. Pieces of context information can be related to one another. For example, with an entity's location, one can determine which other devices or people are near that entity and what kind of activity is occurring near that entity.

4.3.2 Policy-based Context Information Model

The policy-based context information model developed by the author, is designed based on the IETF PCIM (Policy Core Information Model) and its extension, directions which are followed by most of the work carried out in the policy-based management field. The major objective of such information models is to bridge the gap between the human policy administrator who enters the policies and the actual enforcement commands executed at the network elements.

Figure 13 depicts part of the inheritance hierarchy of the information model, representing context in the network-centric approach proposed by the author. Apart from the policy condition and policy action, context information is incorporated into the model by extending it through the addition of the *ContextVariable.* Futhermore, the policy structure used is a trimmed version of the IETF PCIM model. This is done in order to make the policy processing faster, since the size of policies is considerably reduced and their

processing is fast and simpler than the IETF Policy Core Information Model (PCIM) [45][30].

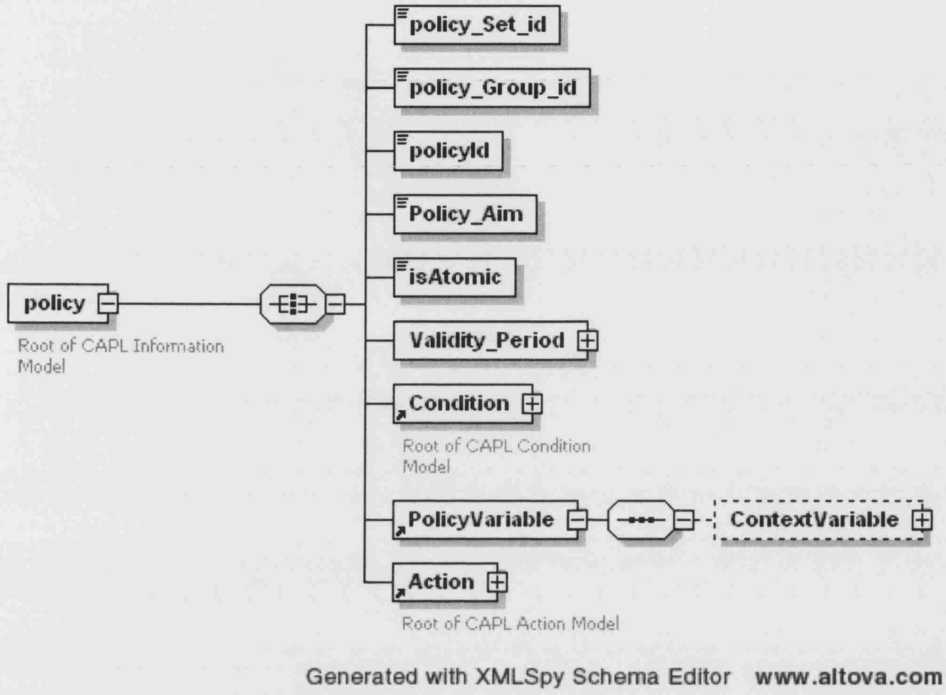


Figure 13: Policy Based Context Information Class Inheritance Hierarchy

Context information is expressed in the policy conditions in the form of elementary Boolean expressions of the form:

```
<Context Variable> MATCH <Context Value>
```

The relationship "MATCH", which is implicit in the model, is interpreted based on the context variable and the context value. Therefore, the modelling of context information starts from *ContextVariable,* which inherits the abstract class available; PolicyVariable.

4.3.3 Context Variable

A variable in general represents information that changes, and it is usually set or evaluated by a kind of software called policy decision point (PDP) in Policy Based

[30] This is due to the fact that the policies will be expressed using XML as it will be described in section 4.4. The smaller the XML file, the faster it can be parsed and then transformed into Java code.

Management terminology. In policy, conditions and actions can abstract information as *PolicyVariable*, or rather *ContextVariable* for context model, to be evaluated in logical expressions, or set by actions.

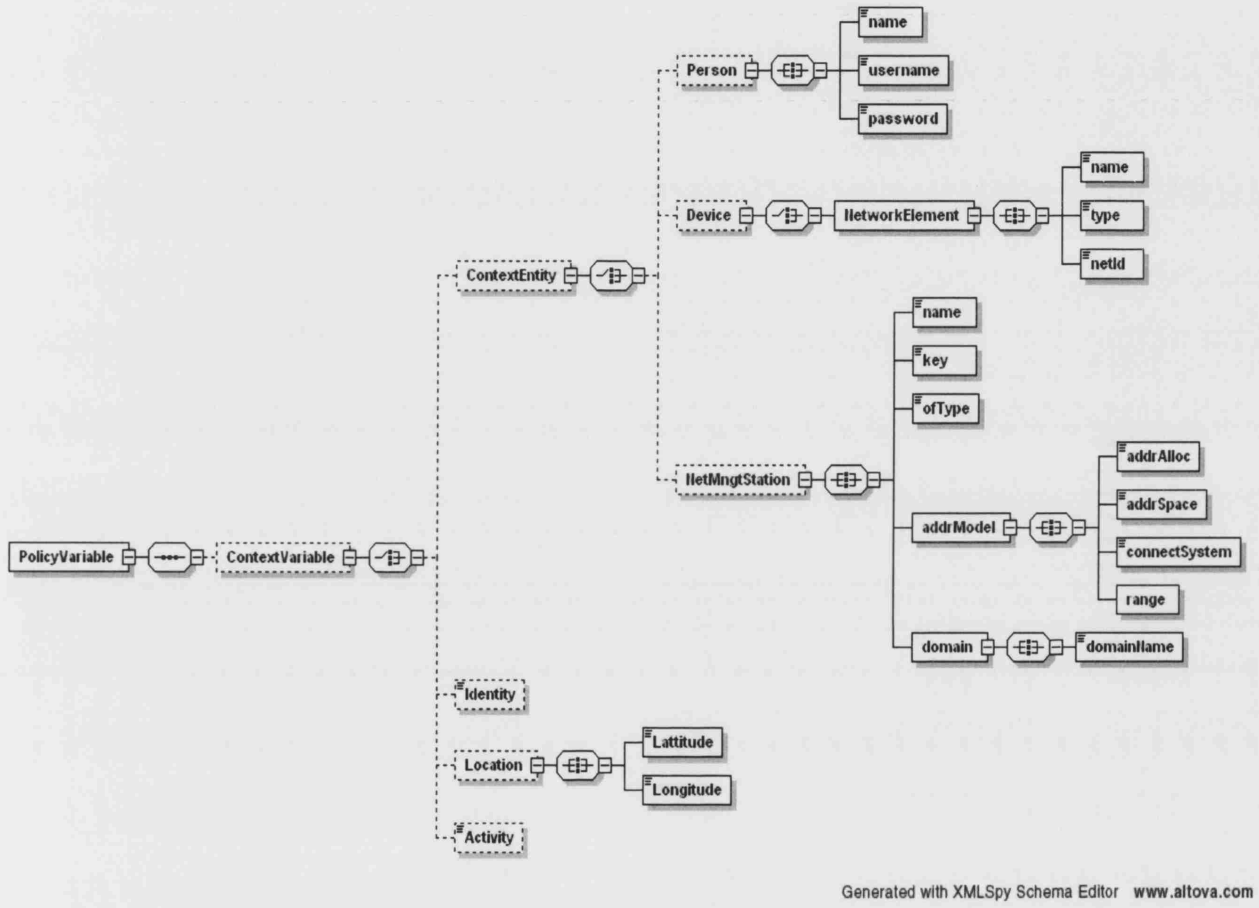


Figure 14: PolicyVariable/ContextVariable Class Inheritance

As discussed in the classification of context section, every context entity should at least be described by four basic features, i.e., identity, location, activity and time. The first three of them are modelled in this class hierarchy as shown in Figure 14.

Since the final fulfillment of a network-centric context-aware service is carried out using the network elements (including mobile handsets), network elements play a vital part in context aware service (CAS) provisioning. As such, *NetworkElement* class is defined explicitly as a child of *Device* class. A NetworkElement can be physical network element

such as a router, switch or virtual network element such as an IP-GRE tunnel, an SNMP agent etc. This can be characterised by:

- Name.

  This is the name by which the device will be known.

- Type.

  This parameter describes the type of network element. It can be an IP-GRE tunnel, an SNMP agent, router, etc.

- netId

  This parameter contains the network identifier of the device. It can be an its IP address, its MAC address, etc..

Network information and intra-/inter-domain information are usually maintained by network management stations at different levels or domains. Therefore, network management station (*NetMngtStation*) is defined as an entity that has all the four basic context types. The detailed description of this class is closely related to the network and service management system. This can be characterised by parameters such as:

- Name

  This is the name by which the device will be known.

- Key

  This is the key used in order to gain access to the resources/services of that network. This can be used to pass the encryption key to be used for joining a network. This could either be the GPRS password or the encryption key used for a wireless LAN.

- OfType

  This is used to describe the type of the network to be joined. The probable values for this one could be anything, like GPRS, WiFi, etc.

- AddrModel

This is used to describe the model of usage of the network. This includes:

- AddrAlloc

  This is used to describe the type of address allocation. This can be either done statically or through the use of the Dynamic Host Configuration Protocol (DHCP).

- AddrSpace

  This is used to describe the type of address space. This can be either private or public.

- ConnectSystem

  This is used to describe how the system is connected. This can be fully masquereded[31] or not masquereded.

- Range

  This is used to describe the range of the address space. This could be any IP address range like 10.0.1.1 – 10.0.1.55.

- Domain

  - DomainName

    This is used in order to define the domain name of the network to be joined. This is the authoritative domain name of the owner of the network.

User and SP information can be easily presented by the *Person* class which is actually the child of *ContextEntity* class therefore no separate class for this purpose is needed.

The Person entity can be characterised as follows:

- Name

  This is used in order to define the name of the user.

[31] This is is also known as *Network Address Translation* (NAT). NAT describes the process of modifying the network addresses contained with datagram headers while they are in transit. IP masquerade is the name given to one type of network address translation that allows all of the hosts on a private network to use the Internet at the price of a single IP address.

- Username

  This is used in order to define the username of the user.

- Password

  This is used in order to define the password of the user.

4.4 Context Aware Policy Language

The Context Aware Policy Language (CAPL) is a language developed in order to express policies, which would include Context information. CAPL is based on XML hence the policies are written in XML[32]. The XML StyleSheet (XS) of the policies used in this prototype is described in the following sections and is the first[33] version of the language.

4.4.1 Overview

Due to the motivation discussed previously[34], the context information is represented as part of the policy, both in high-level policy specification language and low-level OO information model. Policies serve as the containers of context information.

This section presents a first step towards a fully functional context-aware policy definition and processing language (CAPL). It describes the basic concepts, syntax and semantics of CAPL together with the corresponding editing and compiling tools for CAPL. CAPL is a rule-based language that is used by administrators to build up rules for networks and services management by means of defining policies and their processing. By adopting the policy format suggested by IETF Policy Framework Working Group, the policy in CAPL takes the following format:

```
IF <condition(s)> THEN <action(s)>
```

[32] The reasons behind the use of XML for expressing the policies can be found in appendix A.3

[33] The very first version of CAPL (v0.1) can be found in the appendix A.1, together with a simple scenario with which it was tested. This work was presented in the transfer thesis of the author in Sept ember 2004

[34] That both the context-aware service management and the underlying network management are based on the policy based management paradigm

It means *action(s)* is/are taken if the corresponding *condition(s)* is/are true.

A typical context-aware service scenario can be represented by the following policy, which forces the mobile only vibrates rather than beeping during meeting time.

```
IF (location == gower104) and (time within meetingSchedule)
THEN MobileDivert
```

A more network-related example of policy can be simply expressed by the following policy, which means that an IP-GRE tunnel needs to be set up between two ends of communication if the destination host of the communication is within user's enterprise network and the user himself/herself is a manager of this enterprise.

```
IF (destHost within EnterpriseNet) and (userRole is Manager)
THEN useIP-GRETunnel
```

This piece of policy also implies that other traffic would not be carried through the IP-GRE tunnel. By this means, any policies related to the context-aware services can be defined. The execution of these policies eventually triggers the action of classes in policy-based context information model. These rule-based policies are implemented by XML due to XML's built-in syntax check and its portability across the heterogeneous platforms.

4.4.2 CAPL Information Model

As mentioned in the previous sections, there is a need to incorporate context information within the policy management system. A first step towards that direction can be seen in Figure 15.

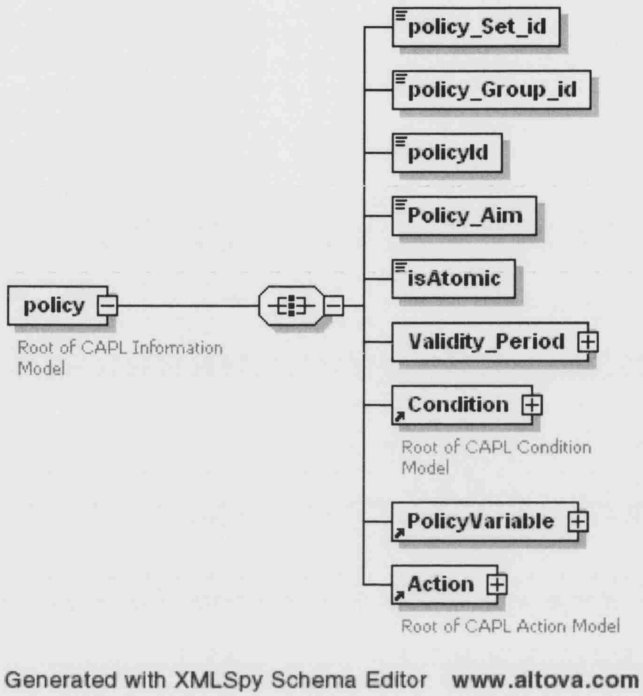


Figure 15: CAPL Policy Information Model

All policies expressed in CAPL, are structured in policy sets and policy groups. The policy sets contain policy groups, which in turn may contain other policy groups or single policies. Keeping this in mind, a policy set can then be seen as a service, whereas the policy group defines the different types of policies, which need to be implemented in order to create and run that service. A detailed example of this can be found in chapter 7, where a service called FollowMe (policySet) contains groups of different types of policies (policy Groups) such as firewall policies, tunneling policies etc. Finally within these groups lie the actual policies implementing the service. From these, it is clear that each single policy will be identified using the vector <Policy_Set_Id, Policy_Group_Id, Policy_Id>. This hierarchical structure will be used to easily manage and organise the storage and processing of the policies loaded in the system.

4.4.2.1 Policy_Set_Id – Policy Set Identification Element

The *Policy_Set_Id* is the first of the identifier elements. The *Policy_Set_Id* element contains the identifier of the policy set to which the policy belongs.

4.4.2.2 Policy_Group_Id – Policy Group Identification Element

The *Policy_Group_Id* field is the next identifier element. It contains the identifier of the group to which this policy belongs.

4.4.2.3 Policy_Id – Policy Identification Number Element

The third identifier element is the *Policy_Id* element. This element contains information which uniquely identifies the policy between the others associated to the same policy Set and policy Group.

4.4.2.4 Policy_Aim – Policy Objective Element

This information is used to describe the policy's aim. This field acts like the policy descriptor so that the system administrator can easily spot the use of the policy.

4.4.2.5 IsAtomic – Atomicity Definition Element

This field, is dedicated to the management of the policy. It defines the way to enforce the policy. The *IsAtomic* element is formed by a boolean value that establishes whether concurrent or atomic execution is followed. If the execution is specified as atomic, the policy must to be enforced before starting the evaluation of the next policy in the sequence.

4.4.2.6 Validity_Period – Validity Period Element

The *Validity_Period* element is intended to express the policy expiration date. Usually the expiration date is just given with the day and hour when the policy starts and finishes. In a future version of the language, filters specifying concrete months, days and hours can be also introduced. Figure 16, depicts the structure of the *Validity_Period* element, from which the only mandatory element is the *Time_Period*, element that includes the start and stop times for the specific policy.

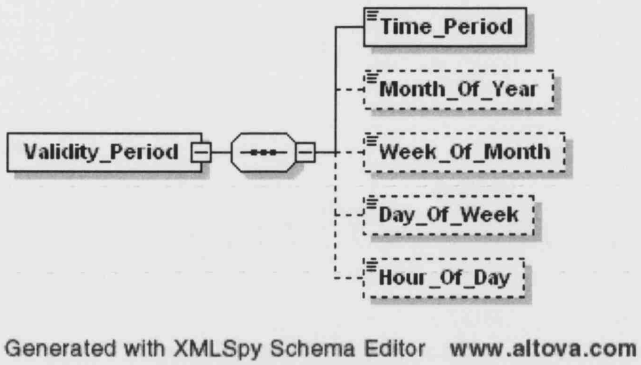


Figure 16: Validity Period

4.4.3 The CAPL Condition

As mentioned previously a policy is expressed in the form 'IF <condition(s)> THEN <action(s)>'. This part of the policy information model describes the condition (Figure 17) part of a policy.

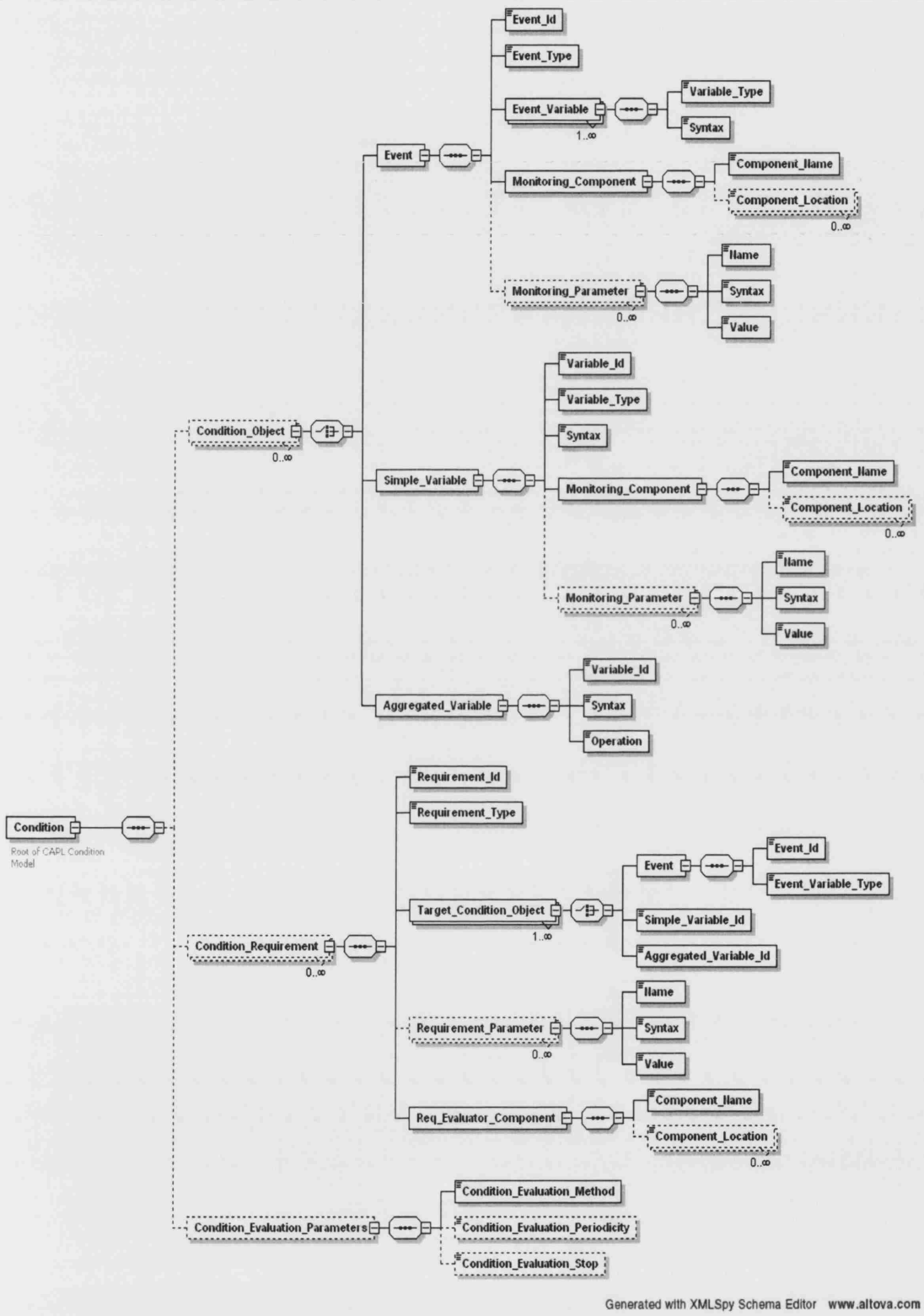


Figure 17: CAPL Condition

The *Condition* element includes all data objects, data requirements and evaluation parameters required to specify and evaluate itself. The *Condition* element contain three basic elements: *Condition_Object, Condition_Requirement* and *Evaluation_Parameters*. The *Condition_Object* (Figure 18) elements specify the data objects that need to be monitored or calculated so that their values can be obtained and used to evaluate and enforce the policy. The *Condition_Requirement* elements contain the different requirements (e.g. thresholds) to be applied over the values of *Condition Objects* in order to decide if the policy condition is fulfilled. The *Evaluation Parameters* includes the evaluation method, the periodicity of the evaluation times and other useful parameters related to the evaluation of the policy condition.

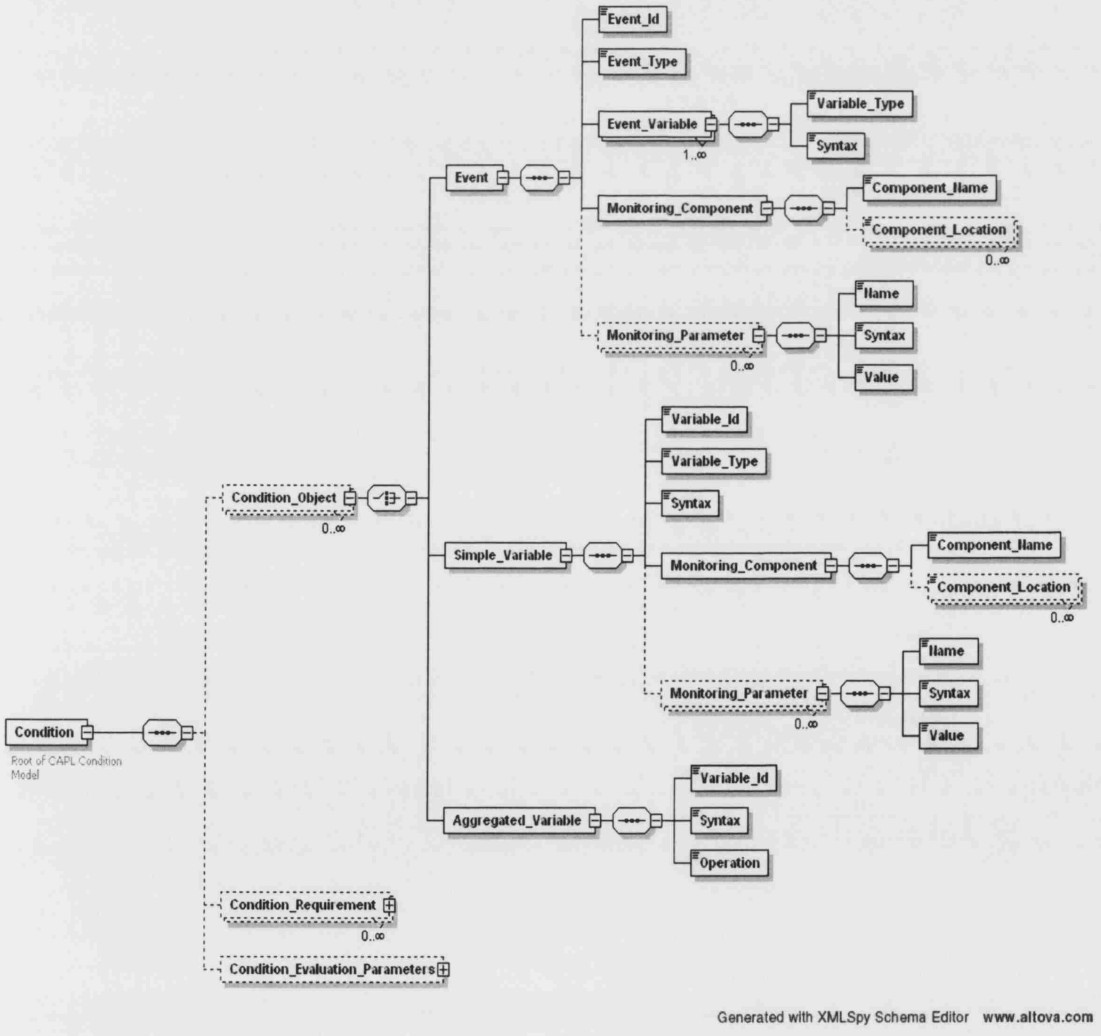


Figure 18: Condition Object

The *Condition_Object* represents the data objects whose values are going to be monitored, and if the *Condition_Requirements* are met, a policy action can be applied and evaluated. The values of the specified *Condition Objects* can also be used as input parameters of the policy actions to be enforced when the policy condition is met. The *Condition Object* element can be of three different types: Events & Event Variables, Simple Variables and Aggregated Variables.

- The *Event* element represents events or occurrences that appear or occur in the external environment (the network) at unpredictable times. The events occur and are not considered as measurable or quantifiable variables at any desired time, just happened at arbitrary times. The appearance of a new user in a coverage area, the reception of a new FTP connection, the appearance of new data in a database, the reception of a new particular message, etc can be thought of as a typical example of this type of element.

  It is considered that an event can contain different pieces of information; this information can be specified using the Event_Variable and is known as such. The values of the event variables can only be obtained when the event occurs. For example, an event reporting the appearance of a new user in a coverage area could contain information about which is the MAC address of the new user in the area, which would be considered to be an event variable. Another example could be an event reporting the appearance of new data in a database, containing the name of the new data appeared as a possible event variable.

  Condition_Requirements can be applied to the values of the *Event_Variables*. Also the values of event variables can be used as input parameter to the policy actions to be enforced when the policy conditions are met.

  The *Event* element contains an *Event_Id* element specifying an identifier of the event (e.g. "Event1") and the *Event_Type* element, specifying the type of event to be monitored (e.g. "new_user_in_coverage_area"). The *Event_Variable* elements are the variables contained in the event. They are identified by the *Type* element (e.g. "new_user_MAC_address") and its syntax type (e.g. integer, float, string, Boolean, etc) by the *Syntax* element. The *Monitoring_Component* element

identifies the particular component (Policy Enforcement Point) that can listen and be aware of the occurrence of the event. The *Monitoring_Component* element contains a *Component_Name* element, the identifier of the software component, and the *Component_Location* element, specifying the place (node) where this component should be installed in order to monitor the event. *Monitoring_Parameter* element is optional and contains any additional information considered useful in reference with the monitoration of the event and that have to be supplied to the monitoring component in order to perform the monitoring activity (e.g. "SSID_name").

- The *Simple_Variable* element represents variables whose values can be monitored and calculated when desired (periodically). These elements are known as 'Simple Variables' The notion of *Simple_Variable* is different from the notion of event and event variable in the sense that these *Simple_Variables* are variables whose values can be continuously monitored and calculated when desired at any specified time.

  It is considered that these variables could be of two kinds: *external environment variables* or *policy deduced variables*. The *external environment variables* are external environment measurable variables at some point over the network representing some kind of characteristic or state. Examples of this kind of variable could be the delay at some link, the load or lost packets at some router interface, the number of connections opened to some server, the local time of a node, etc.

  The *policy deduced variables* can be understood as high level variables defined at policy level. The values of these variables can be modified as a result of some policy action and can be involved in the evaluation of other policies. This kind of variable is the chain to allow interleaved policies. This kind of variables only exists at policy level and will be calculated, stored and managed internally by the Policy Based Management System. An example of this kind of variable, could be the severity level maintained during the service assurance phase, because this variable is not directly measured in the nodes of the network, but is a high level

state variable, which value can be assigned or modified due to the action enforcement of different policies.

The *Simple_Variable* elements contain a *Variable_Id*, *Variable_Type* and *Syntax* elements that specify an identifier (e.g. "Var1"), the type of variable to be monitored (e.g. "PacketLoss") and its syntax type (e.g. "integer"). The *Monitoring_Component* element identifies the component to be used to monitor the simple variable. A Policy Enforcement Point will monitor the external environment variables. The policy deduced variables will be monitored by the component responsible for making decisions (Decision Maker Component - to be described in the following sections). The *Monitoring_Parameter* element is optional and contains additional information in reference to the monitoring of the simple variable.

- The *Aggregated_Variable* element represents variables whose values are obtained after applying some arithmetic operations over the values of one or more simple variables or event variables (that must be previously specified using *Simple_Variable* element or *Event* element). This element contains an *Identifier* of the aggregated variable, its *Syntax* (e.g. "integer"), and the *Operation* to perform over the values of other variables in order to obtain the aggregated variable. The *Operation* element contains a list of operations and the parameters required to perform them. The format of the *Operation* element can be a string containing identifiers of declared event variable and simple variables and operator symbols. For example, the content of the operation could be "(Variable1 + Variable2) / Variable3". A calculation mechanism for parsing and evaluating this operation is included in the logic of the Decision Maker component.

The *Condition_Requirement* elements (Figure 19) contain the requirements that must be applied over the values of *Condition_Objects* specified so that the condition can be evaluated and considered as achieved or not. Note that a policy condition can be composed by more than one Condition Requirement. Each and every Condition Requirement has to be independently evaluated (Requirement Evaluation). Depending on

the results of the different requirement evaluations, the policy condition will considered fulfilled or not (Condition Evaluation).

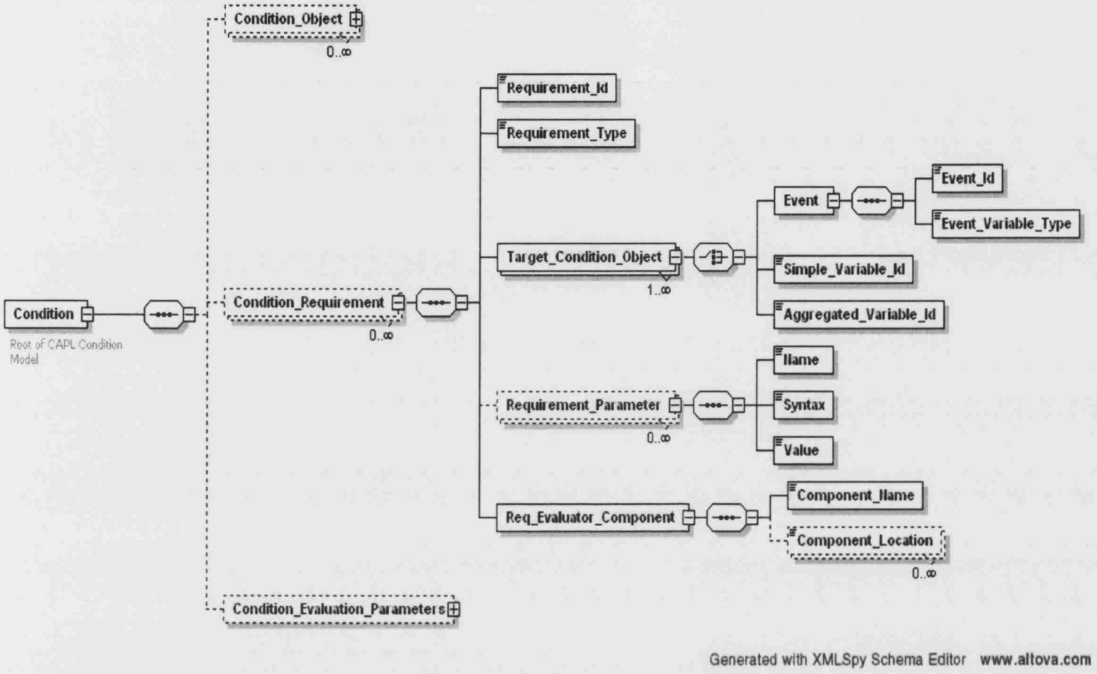


Figure 19: Condition Requirement

A *Condition_Requirement* is identified by the *Requirement_id* element. The *Requirement_Type* element must specify the type of requirement to be applied and evaluated, as for instance, matching some value (Var_1 = X), lower than some minimum threshold (Var_1< Y), etc. The basic types of requirements that can be applied are:

- *Max_Threshold*: if the value of the variable, for which the requirement applies, is greater than a threshold value then the requirement is fulfilled. The name of the variable to be evaluated must be specified within a *Target_Condition_Object* element. The value of the threshold must be supplied as a *Requirement_Parameter*. The name of this *Requirement_Parameter* must be "Max_Threshold".
- *Min_Threshold:* if the value of the variable, for which the requirement applies, is lower than a threshold value then the requirement is fulfilled. The name of the variable to be evaluated must be specified within a *Target_Condition_Object*

element. The value of the threshold must be supplied as a *Requirement_Parameter*. The name of this *Requirement_Parameter* must be "Min_Threshold".

- *Match_value:* if the value of the variable, for which the requirement applies, is equal to the specific value then the requirement is fulfilled. The name of the variable to be evaluated must be specified within a *Target_Condition_Object* element. The specific value must be supplied as a *Requirement_Parameter* whose name must be "Match_Value".
- *In_Margins:* if the value of the variable, for which the requirement applies, is within a margin of values then the requirement is fulfilled. The name of the variable to be evaluated must be specified within a *Target_Condition_Object* element. A high edge value and a low edge value must be supplied as *Requirement_Parameters*. Their names must be "High_Margin" and "Low_Margin" respectively.
- *Out_Margins:* if the value of the variable, for which the requirement applies, is outside a range of values then the requirement is fulfilled. The name of the variable to be evaluated must be specified within a *Target_Condition_Object* element. A high edge value and a low edge value must be supplied as *Requirement_Parameters*. Their names must be "High_Margin" and "Low_Margin" respectively.

The variables to be checked against the requirements identified are specified inside *Target_Condition_Object* elements by their names, differentiating by each other through the use of event variables, simple variables or aggregated variables. The *Requirement_Parameter* elements specify the needed parameters in order to apply the requirement, and are specified by *Name*, *Syntax* and a *Value*. These parameters are specific to the type of requirement (for instance, if a requirement contains a minimum threshold; the *Requirement_Parameter* must contain the value of this threshold). The *Req_Evaluator_Component* element identifies the component responsible for applying the requirement, and performing the requirement evaluation. This component can be a Policy Enforcement Point or the Decision Maker Component. The Requirement

evaluation should be carried out as near to the Policy Enforcement Point as possible in order to minimise the information exchanged between the Decision Making Component and the Policy Enforcement Point. If the Condition Objects in a requirement are all monitored by a same Policy Enforcement Point, the requirement evaluation should be performed by the Policy Enforcement Point itself. The place where the component is installed can be specified in *Component_Location* element.

The *Evaluation_Parameters* (Figure 20) element contains information regarding the evaluation process.

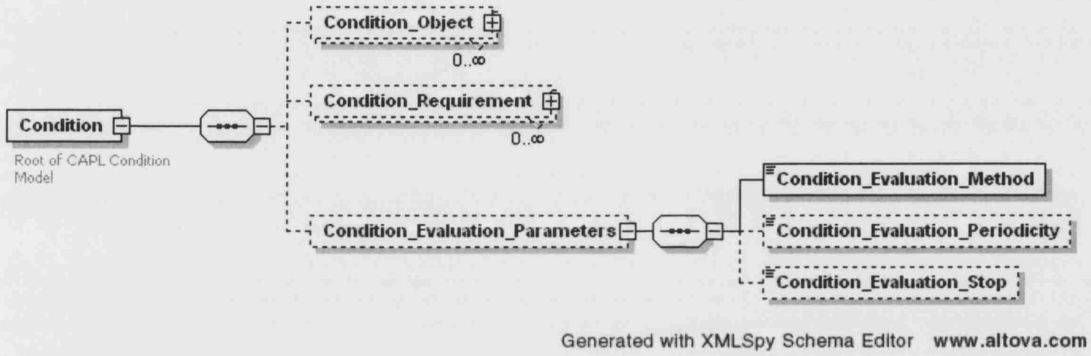


Figure 20: Condition Evaluation

The *Evaluation_Method* element describes how the *Condition Requirements* specified must be satisfied in order to consider the overall condition fulfilled. This element can specify if all the requirements identified have to be fulfilled in order to consider the condition also fulfilled, or only one of them, etc. This element must contain the *Condition Requirements* identifiers (*Requirement_Id*) and logical operators (AND, OR, NOT). For example, the content of this element can be "(Requirement1 AND Requirement2) OR Requirement3". The *Evaluation_Method* specifies how to perform the condition evaluation and when to decide if the policy condition is fulfilled. The condition evaluation will be always performed by the Decision Maker Component. The period of time for the collection and condition evaluation process (for variables requiring time driven measurements) is contained in the *Evaluation_Periodicity* element (optional).

The *Condition_Evaluation_Stop* establishes when the evaluation process will be automatically stopped. For instance, it could be specified within this parameter that after having met the condition once, or X times, or after some period of time, the evaluation of the policy will be stopped (this will imply that the monitoring of those variables will also be stopped). If this element is not specified, the policy evaluation will continue until the policy expires, or be removed from the system, or stopped as a result of the enforcement of an action.

Finally, it is important to note that the elements contained within *Condition* element are optional (*Condition_Object, Condition_Requirement,* and *Condition_Evaluation_Parameters*). This is because a policy without any condition to evaluate is also a valid policy. Such a policy has to be enforced immediately without any previous evaluation (it can be considered that the condition is always true).

4.4.4 The CAPL Action

The *Action* elements contain all the information needed for the enforcement of the specific action. The structure of the Action element is shown in Figure 21. The *Action_Id* field within the *Action* element contains an identifier of the action to be enforced (e.g. "Action1"). The *Action_Type* element specifies which specific action has to be enforced (e.g. "create_route").

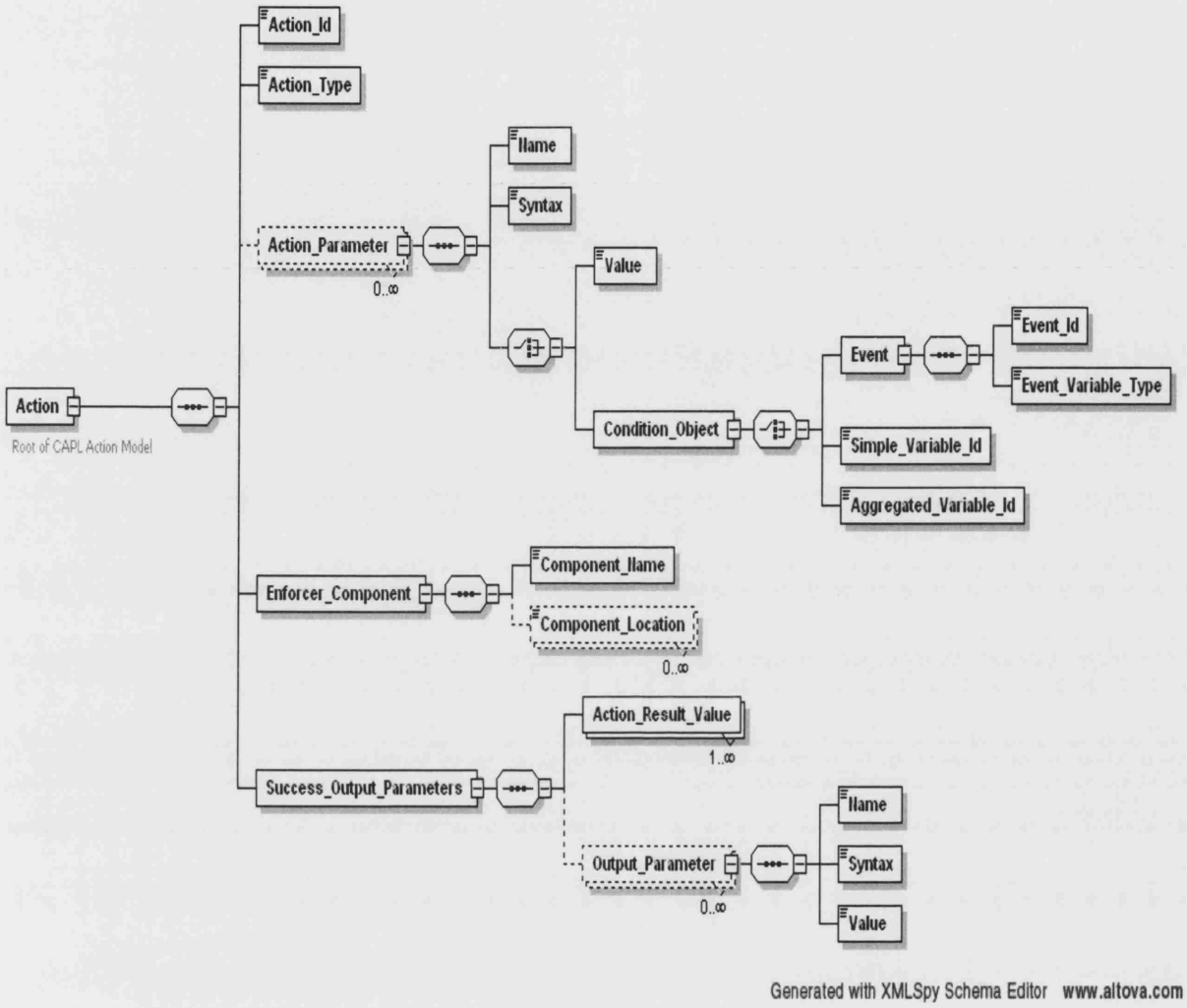


Figure 21: CAPL Action

The *Action_Parameter* element is an optional element that contains a parameter to be applied during the action enforcement. These parameters are action specific and its values are used when the action has to be enforced. Each action parameter is identified by a *Name* element. The *Syntax* element specifies the syntax type of the value of the parameter. The content or value of the action parameter can be specified either supplying a static value defined at policy creation/edition time (*Value* element) or assigning the value of a Condition Object (*Condition_Object* element). If a Condition Object is associated to an action parameter, when the action has to be enforced, the value assigned to the action parameter will be the current value of the Condition Object specified.

The *Enforcer_Component* element defines the component responsible for enforcing the policy action. The possible components responsible for enforcing an action are either a Policy Enforcement Point or the Policy Manager. The Policy Enforcement Points are responsible for complex actions enforcement over the external environment. The Policy Manager is responsible for the enforcement of simple actions regarding the assignment of values to policy deduced simple variables, or actions demanding the activation or deactivation of other policies loaded in the system. The *Component_Name* identifies the particular Policy Enforcement Point or the Policy Manager. The place (address) where the enforcer consumer is installed (addressable) can be specified by the *Component_Location* element.

The *Enforcement_TimeOut* element specifies the maximum time available to enforce the action before considering the enforcement as failed. *Success_Output_Parameters* contains expected results from policy actions after having been enforced to consider the action successfully enforced. The *Action_Results_Value* elements include the expected values of the action results in order to consider the action successfully enforced, and optionally *Output_Parameter* elements specific of the action, with its expected values. This information will be used by the Policy Manager to decide if the policy has been enforced successfully or not.

4.5 Policy Storage Service

The Policy Storage Service (PSS) is the place where all the policies are stored. This is a PostgreSQL database [99], offering a well-defined interface to the other system components. The interface was designed and built in order to offer the following methods to the Policy Management Tool (described in the section 4.8) as well as the Policy Manager in order for them to communicate with the database:

- *'addPolicy'* . This method provides the mechanism to add a new policy to the policy repository. The input parameters used are the policySetId, the policyGroupId, policyId and the policy. The policySetId is the identifier of the policySet to which the policyGroup belongs or to which the policy to be added directly belongs. The policyGroupId is the identifier of the policyGroup to which

the Policy to be added belongs (if any). The policyId is the identifier of the policy to be introduced. The policy is the CAPL expressed policy. This method issues an SQL query of the form: *'INSERT INTO caplf2 VALUES (policySetId, policyGroupId, policyId, policyXML, policyJava);'*

- 'addPolicyGroup'. This method provides the mechanism to add a new policy group to the policy repository. Input parameters: policySetId: identifier of the policySet to which the policyGroup belongs. (It should be previously created in the repository, using 'addPolicySet', before attempting to introduce a policy).policyGroupId: identifier of the policyGroup to be introduced.policyGroup: the object containing the information regarding the policyGroup we want to introduce. This method issues an SQL query of the form: *'INSERT INTO policyGroupT VALUES (policySetId, policyGroupId, policyGroupXML);'*
- 'addPolicySet' method provides the mechanism to add a new policy set to the policy repository. Input parameters: policySetId: identifier of the policySet we want to introduce.policySet: the object containing the information regarding the policySet we want to introduce. This method issues an SQL query of the form: *'INSERT INTO policySetT VALUES (policySetId, policySetXML);'*
- *'retrievePolicy'* method provides the mechanism to retrieve a specific policy from the policy repository. It can also be used to retrieve a set of policies belonging to a specific policy group or a specific policy set. Input parameters: policySetId: identifier of the policySet to which the policyGroup belongs, or to which the policy to be retrieved directly belongs. policyGroupId: identifier of the policyGroup to which the Policy required to be retrieved belongs to (if any). policyId: identifier of the policy to be retrieved. This method issues an SQL query of the form: *'SELECT policy FROM caplf2 WHERE (policySetId=policySetId1 AND policyGroupId=policyGroupId1 AND policyId=policyId1);'*
- *'retrievePolicyGroup'* method provides the mechanism to retrieve a specific policy group information from the policy repository. Input parameters: policySetId: identifier of the policySet to which the policyGroup belongs. policyGroupId: identifier of the policyGroup to be retrieved. This method issues

an SQL query of the form: *'SELECT policyGroupT FROM policy WHERE (policySetId=policySetId1 AND policyGroupId=policyGroupId1);'*

- *'retrievePolicySet'* method provides the mechanism to retrieve a specific policy set information from the policy repository. Input parameters: policySetId: identifier of the policySet to be retrieved. This method issues an SQL query of the form: *'SELECT policy FROM policySetT WHERE (policySetId=policySetId1);'*
- *'removePolicy'* method provides the mechanism to remove a policy from the policy repository. It can also be used to remove a specific policy group or a specific policy set. Input parameters: policySetId: identifier of the policySet to which the policyGroup belongs, or to which the policy to be removed directly belongs. policyGroupId: identifier of the policyGroup to which the Policy we want to remove belongs (if any). policyId: identifier of the policy to be removed. If the policyId parameter is null, the method will remove the policyGroup specified (and all its belonging policies). If the policyGroupId is also null, the method will remove the policySet specified (and all the policies and policyGroups belonging to it). This method issues an SQL query of the form:
    - 'DELETE FROM caplf2 WHERE *((policySetId=policySetId1 AND policyGroupId=policyGroupId1 AND policyId=policyId1));'*
    - 'DELETE FROM policyGroupT WHERE *((policySetId=policySetId1 AND policyGroupId=policyGroupId1));'*
    - 'DELETE FROM policySetT WHERE *((policySetId=policySetId1));'*

4.6 The CAPL Policy Manager

The CAPL Policy Manager (CAPLPM or PM). This component is responsible for deciding which policies are applicable when a set of given conditions are met. The Policy Manager (CAPLPM) is always waiting for new policies to arrive from the Policy Management Tool (PMT). The specific fields required for these policies and its general structure have already been described in section 4.4.

Once a new policy, or policies, arrives as an XML document, from the Policy Management Tool, the PM parses it to Java and sends it to the Policy Conflict Checker

Component in order to check for conflicts with other policies that are already installed in the system. In case of conflict detection, the policy will be rejected.

Once all the checks have been completed, the PM starts processing the different policies received in order to decide how to proceed. The PM checks which of them have to be activated immediately, or which of them will be stored for future activation. Policy activation occurs when the policy conditions are sent to the Decision Maker Component in order to start the evaluation process. The evaluation process includes the monitoring of the specified variables (in the policy condition). If the condition becomes true, the Decision Maker Component will notify the PM. If the policy is not activated immediately, it will be stored in the Policy Storage Service (PSS) for further processing. In that case the policy will not be evaluated until its set time.

The policies are structured hierarchically. A root element Policy is located at the top. From the root element, Policy Sets are defined which contain Policy Groups. A Policy Group can contain single Policies. The Policies contained in Policy Groups are ordered following some sequence. This sequence is maintained and used by the PM to decide which Policies will be activated at any time. The first Policy in the sequence inside the first Policy Group will be activated first. The second policy in the sequence will be activated either at the same time or after the first Policy is enforced. If the first Policy is labelled as an Atomic one, then the second Policy will only be activated if the first one has been enforced. If the first Policy is labelled as a not Atomic one, then the second Policy can be activated simultaneously with the first one. The different Policy Groups inside a Policy Set are also ordered with a sequence number. The first Policy Group in the sequence will be processed before the second one. The Policy Groups can be also labelled as Atomic or not. If the Policy Group is Atomic, then all the Policies contained have to be previously enforced before activating the Policies of the second Policy Group. The Policy Manager treats each of the Policy Sets independently.

If the Policy Manager decides that a Policy has to be activated, the Policy condition will be sent to the Decision Maker to be evaluated. Once this is done, the monitoring of the required variables will commence. The policy condition will be sent from the Policy Manager to the Decision Maker in Java serialized objects.

When the Decision Maker decides that the condition of a Policy is met, it notifies the Policy Manager to retrieve the actions associated with this Policy from the Policy Storage Service, and to send them to the appropriate Policy Enforcement Points in order to enforce them. Not all policies are enforced by the Policy Enforcement Points. Some Policies, whose actions result in the modification of local policy variables or to the activation/deactivation of other Policies loaded in the system, are enforced by the Policy Manager itself.

4.7 The CAPL Decision Maker

The CAPL Decision Maker (CAPLDM) receives the policy conditions from the Policy Manager and is responsible for the condition evaluation process. Once this is successful a notification is sent to the Policy Manager announcing that the condition was met. So the CAPLDM is the component directly related to the evaluation process of the policy conditions. When the CAPLDM decides that the condition of a policy is met, the PM will send to the appropriate Policy Enforcement Points the policy actions associated with the condition met.

It is very important to explain the concept of condition evaluation. The condition of a policy contains three fundamental components: the *Condition Objects*, the *Condition Requirements* and the *Evaluation Method*. One *Condition Requirement* (a threshold, a match value, margins, comparisons, etc) is applied over the values of one *Condition Object* (*Condition Objects* can be events, simple variables or aggregated variables). A condition is composed by one or more *Condition Requirements*. If more than one *Condition Requirements* are specified in the condition, then a *Evaluation Method* to evaluate the condition must be supplied, e.g. (Requirement1 AND Requirement2) OR (Requirement3). The condition evaluation process refers to the application of the *Evaluation Method* in order to decide if a complex condition is met. Requirement evaluation on the other hand refers to the application of one single *Condition Requirement* in order to evaluate if it is met.

During the evaluation process three main activities must be performed: Obtain the values of the *Condition Objects*, apply the *Condition Requirements* (requirement evaluation), and apply the *Evaluation Method* (condition evaluation).

The CAPLDM will use the Policy Enforcement Points (PEPs) in order to support this evaluation process. The PEPs are software components that can be installed in different points of the network and can perform monitoring and requirement evaluation activities. Both CAPLDM and PEPs can perform evaluation activities. So PEPs can have the responsibility for evaluating requirements (apply *Condition Requirements*) if all the events or variables (*Condition Objects*) needed are handled by the same PEP (this could reduce the amount of information transferred between the CAPLDM and PEP, because only when the values of the variables or events match the requirements imposed by the condition, the PEP will inform the CAPLDM).

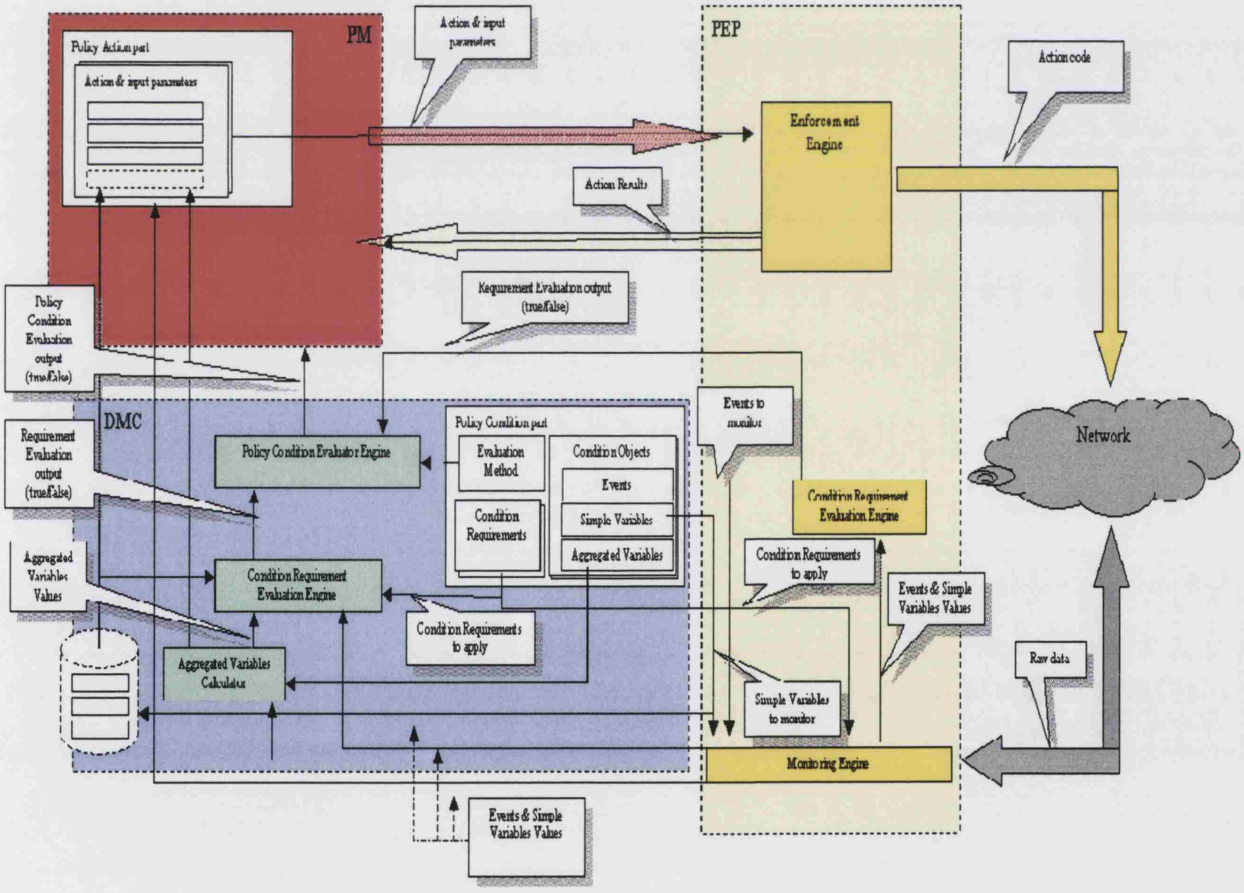


Figure 22: Policy Condition Evaluation

If the parameters needed to evaluate some requirement are monitored by different PEPs, the component responsible for this aggregation of information and requirement evaluation is the CAPLDM. In this case, the PEP will only perform monitoring tasks, informing the CAPLDM periodically of monitored variables values. The condition evaluation (applying the *Evaluation Method*) will always be the responsibility of the CAPLDM.

Figure 22 shows the Policy condition process and the functional components involved. The figure tries to clarify the responsibilities of the CAPLDM and the PEPs for performing each of the main activities involved during the evaluation process.

- *Condition Objects monitoring*: The Condition Objects can be defined as Events, Simple Variables or Aggregated Variables.

Events elements are events or occurrences that appear or occur in the external environment (the network) at unpredictable times. Those events are not considered as measurable or quantifiable variables at any desired time, they just happen at unspecified times. A Policy Enforcement Point will monitor the Events and the Event Variables. So, the monitoring engine of the PEP is responsible for monitoring this type of event.

The *Simple_Variable* elements represent variables whose values can be monitored and calculated when desired (periodically). This type of variable will be measured by a PEP as well.

The *policy deduced variables* are high level variables defined at policy level. The values of these variables can be modified as a result of some policy action and can be involved in the evaluation of other policies. This kind of variable only exists at policy level and will be calculated, stored and managed locally by the Policy Based Management System. The values of this kind of variable will be obtained by the CAPLDM as this kind of variables only exist locally and will be stored internally in the PBMS.

The *Aggregated_Variable* element represents variables whose values are obtained after applying some arithmetic operations over the values of one or more simple variables or event variable. The CAPLDM is the component responsible for calculating the values of this kind of variable. In this case, the values of the Simple or Event Variables are

calculated at PEP level and then forwarded to the CAPLDM for aggregation (Aggregated Variable).

- *Condition Requirements evaluation*: Depending on how the condition is composed, the DM and PEPs can play different roles.
    1) If the condition is composed by one Condition Requirement over a Condition Object.
        a) If the Condition Object is an Event to be monitored by a PEP, it is more efficient that the Condition Requirement is evaluated by the PEP itself (Condition Requirement Evaluation Engine). So if the requirement is met, the PEP will send a notification to the CAPLDM announcing that the Condition Requirement is met. So the task of the CAPLDM in that case is trivial, the condition is met.
        b) If the Condition Object is a Simple Variable to be monitored by a PEP (external environment variable), the Condition Requirement is evaluated by the PEP itself. Therefore, if the requirement is met, the PEP will send a notification to the CAPLDM announcing that the Condition Requirement is met. So the task of the CAPLDM in that case is trivial, the condition is met.
        c) If the Condition Object is a Simple Variable to be obtained by the CAPLDM (policy deduced variable), the Condition Requirement is evaluated by the CAPLDM. If the Condition Requirement is met then the CAPLDM itself will decide if the condition is met.
        d) If the Condition Object is an Aggregated Variable to be calculated by the CAPLDM, the Condition Requirement will be evaluated by the CAPLDM. If the Condition Requirement is met then the CAPLDM itself will decide that the condition is met.
    2) If the policy condition is composed by more than one Condition Requirements to be applied over the values of different Condition

Objects, each of the Condition Requirements will be processed as explained previously.

- *Condition evaluation*: In order to decide if an overall policy condition is met, the CAPLDM will apply the Evaluation Method expressed in the policy condition (e.g. (Requirement1 AND Requirement2) OR (Requirement3)) in order to do it. Obviously the Evaluation Methodology will only make sense in the case that the policy condition is a complex one, i.e. is composed of more than one Condition Requirement.

Note that the values of the Condition Objects can not only be used to evaluate the policy condition but also to specify input parameters of the policy actions to be enforced.
The PEPs can be installed in active nodes or not. The CAPLDM will send a message to the PEP containing the variables (Condition Objects) to be monitored, and the possible constraints that its values must meet (Condition Requirements) to be checked. This message will also include an identifier. The PEP, having received this information, will start the monitoring and evaluation process. Every time one or more of the requirements is met, a message with the same identifier received will be sent back in reply.

4.8 Policy Management Tool

The Policy Management Tool (PMT) is the user interface for inserting policies. This is using the interface of the PSS in order to store the policies into the system. It currently supports all the methods described in the Policy Storage Service (PSS).

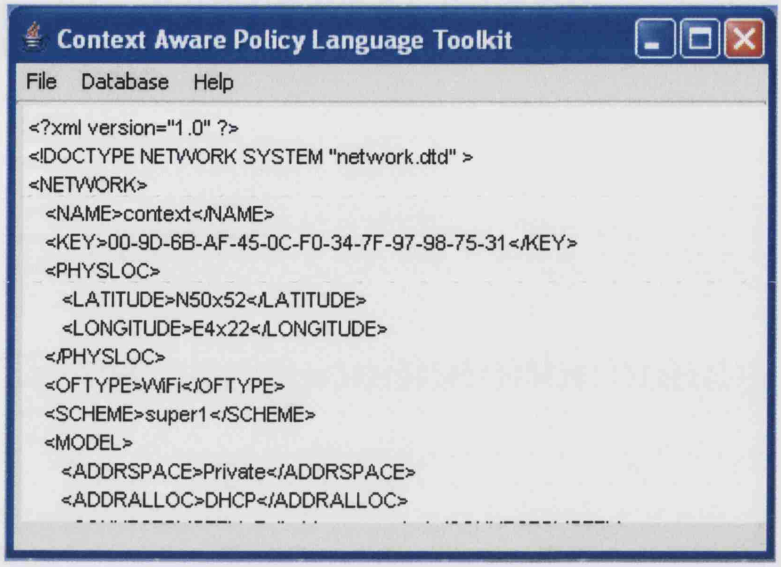


Figure 23: CAPL Policy Management Tool

In Figure 23, the Policy Editor component of the PMT can be seen. This is the area where the policies are composed[35]. Once the policies are authored, the administrator from the File menu can select to '*Insert policy*'. By doing so, the policy is sent over to the Policy Manager for parsing and compilation.

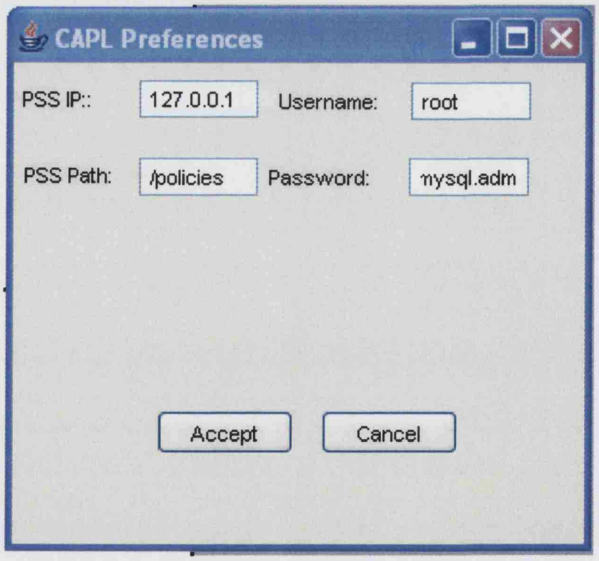


Figure 24: CAPL Preferences

[35] Authoring policies is straightforward; the policy author selects an XML policy template to fill in.

Once that is done, the policy is passed on to the Conflict Checker. If the new policy conflicts with any of the existing ones, then the policy is rejected, and the administrator gets a message indicating so.

In Figure 24 the CAPL preferences dialogue box can be seen. Through this part of the CAPL PMT the administrator can set up the properties of the PMT so that it can locate the Policy Storage Service. Fields, which have to be filled in by the administrator before carrying on authoring policies, include:

- PSS IP

  This field contains the IP address where the PSS can be found.
- Username

  This field carried the username of the user in order to access the specific PSS
- PSS Path

  This field contains the path where the policies are stored in the PSS database
- Password

  This field contains the administrative password of the user in order to access the specific PSS

Once this information is entered, it is stored in the system so it can be reused.

4.9 Conflict Checker

The Conflict Checker (CC) is responsible for identifying conflicts between policies. The policy model has opened up the possibility of systematic analysis of policy conflicts. This can be done by recognising the overlap of policy condition objects between different policies is a necessary condition for a conflict. If there are no objects in common between two policies then there is no possibility of conflict.

Conflicts of modalities can be recognised independently of any specific application. They assume overlaps between conditions, actions of the two policies:

- There may be a direct contradiction of modalities, i.e. two simultaneous policies, one positive permitting and one negative, prohibiting the same action on the same to be performed.

- There may be an inconsistency between motivation and authorisation. When one is motivated to perform an operation but not authorised to do so.

Furthermore, policy conflicts, which are dependent upon the application, can occur:

- Enforcement Points overlap.

  When the enforcement points of two policies overlap and same conditions result in different actions, there is a potential for conflict.

Once the new policy arrives to the Policy Manager, it is forwarded to the Conflict Checker. The Conflict Checker then retrieves all the policies, which are currently installed in the system, and compares them to the newly authored one. If a potential conflict is identified, a notification is sent to the policy author, who would then decide if the policy is to be installed or not.

4.10 Policy Enforcement Points

The Policy Enforcement Points are responsible for implementing the policy actions. They have two main functionalities:

- The Condition Evaluation
- The Condition Monitoring

The Policy Enforcement Points are intended to monitor and evaluate conditions regarding specific services. The conditions can refer to monitoring events that must trigger some code actions. These components are policy/service specialised, i.e. they can monitor and enforce the policies related to a specific service.

The PEPs communicate their information to the Policy Manager and the Decision Maker through the use of messages. Specialised PEPs have been implemented to support the scenarios in chapter 7 (7.2.8.1 and 7.3.8.1).

4.11 Conclusions and Summary

This section described a Policy Based Management methodology to tackle the context-awareness in the field of network and service provisioning and management (rather than the traditional HCI field) by means of context modelling. This object-oriented context

model as proposed in this section is based on the IETF policy core information model and its extensions, therefore making it consistent with the whole lifecycle of context-aware service provisioning.

From scarce availability of context modelling, this section contributes to the current state-of-the-art by presenting an object-oriented context information model for network-centric context-aware services. Furthermore, this context model takes into consideration the service management and underlying network management by following the same methodology, which is the policy-based paradigm.

Furthermore, the first version of CAPL was developed and presented. CAPL is a policy language[36] generated by the author, in order to express policies, which include context information. CAPL is based on XML hence the policies are written in XML. Apart from CAPL, a Policy Based Management Framework was developed in order to be able to use the context information within CAPL and demonstrate its use. This framework consists of:

- A policy repository for storing the policies
  This was developed together with an interface to it, providing a way to access, store, modify and remove policies from it.
- A Conflict Checker component
  This component is responsible for identifying potential conflicts, and notifying the policy author.
- A Policy Manager Component
  This component is responsible for handling the policies.
- A Decision Making Component
  This component is responsible for making decisions regarding the active policies.
- A basic graphical user interface for the CAPL framework
  Through this the administrator can control the Policy Based Management System. The administrator can add, modify, remove and view policies in the system and modify the configuration the Policy Storage Service
- A simple PEP

[36] CAPL follows the IETF policy definition as described in sections 1.5 and 1.6

In this chapter a general description of a PEP, was presented. Chapter 6, uses a dummy PEP[37] just for testing purposes, whereas in chapter 7, two PEPs are implemented in order to support the scenarios devised.

In chapter 5, the technologies used in order to develop the CAPL Framework will be presented, and this will be followed by the functional testing of each of the components in chapter 6. Finally the Policy Framework developed will be exercised in an integrated fashion in chapter 7.

[37] This is a dummy PEP which will be customised for supporting the monitoring and action enforcement required by a particular service

5 Enabling Technologies

5.1 Introduction

This chapter presents the different technologies used in order to implement and eventually test the CAPL Framework. Some of the technologies presented in this chapter will be used in chapter 7, in order to build some test services, through which the use of the CAPL Framework will be demonstrated. These technologies include:

- An Active Platform for distributing, applying and monitoring the policies
- A set of Linux routers
- One network and one service adaptation (WLAN and SIP).

5.2 Active and programmable networks

Many conceptual ideas for programmable networks have been proposed recently. Depending on the literature, the definition of the programmable network varies. For example Campbell et al. [106] defines two schools of thought that have emerged on how to make a network programmable. The first one is spearheaded by the Opensig community that was established through a series of international workshops [107], and the other, established by DARPA [108], constitutes a large number of diverse Active Network projects. Similar definition can be found from the articles by Kalaiarul Dharmalingam and Martin Collier [109], where network programmability has been divided into the three categories: the Opensig initiative, Active Networks (AN), and Mobile Agents. In the following sections, the AN concept is described in more detail.

ANs are packet-switched networks where packets can carry not only data but also a suitable code portion/reference that will be executed at intermediate nodes as they propagate through the network.

In the traditional network, each node performs only the processing necessary to forward packets towards their destination. Instead, an AN is aware of the content of the packets, which are flowing through it, and is capable of making customised modifications to the

data within the packets. In other words, while traditional, 'passive' network function is store-and-forward, the AN is store-compute-and-forward.

The AN's capability to modify traffic inside it, is based on active routers and switches which are able to perform customised computation on the messages streaming through them. Active network components can not only work in the network formed purely from active components but those can also coexist and interoperate with legacy routers, which transparently forward datagrams between the active parts of the network.

There exist two main approaches to realise active networks: *Programmable Node* and *Encapsulation.* In the first solution, programs are injected into the programmable active node separately from the actual packet. This approach uses existing network packet formats and provides a discrete mechanism for downloading programs to the active nodes. User first stores the program to the network node. The program is executed when data packets associated with it arrive at the node [110][114].

In contrast, in the *capsule* approach a program is integrated into every packet. Encapsulation leaves present packet formats behind and replaces them with midget programs that are encapsulated within the transmission frames. When a capsule arrives at the node, an Execution Environment interprets the program and executes it. In this approach, the active node has a built-in mechanism to load the encapsulated code, an execution environment to execute the code and a relative permanent storage where capsules can retrieve or store information [110][114].

Regardless of the approach, the main idea stays the same, the AN customises its content and the behaviour of the network can be altered dynamically by injecting new programs to the network nodes. The ability to inject software to the nodes gains benefits such as rapid application and protocol deployment, software updates and customisation by downloading user specific programs that are downloaded only when needed.

In order to test the functionality devised, it is important that the policies are somehow deployed into the network. To achieve the efficiency in the deployment, delivery, and utilisation of these context-aware policies, the rapid and adaptive service execution platform is needed. In addition to these requirements, the developed platform should be network infrastructure independent, span over a number of heterogeneous network technologies.

A solution to these requirements, the use of programmable network technology on top of fixed and mobile networks, was selected. The author believes that next-generation service networks can benefit from the programmable networking paradigm with the dynamic deployment of network services that can be tailored to the user's requirements using context.

The essential feature of Active Networks is the programmability of its infrastructure on demand. This characteristic creates a versatile network that can easily adapt to the future needs of applications. This will be used to distribute the policies to the network.

Applications have to deliver information to users over a variety of access technologies from optical fibre to copper cables and even wireless transmission. Applications need to be network-aware to adapt to the constraints of the underlying network hardware. Another point is that the mobile devices have to be treated as static devices because it is not currently possible to move a mobile phone seamlessly across the network.

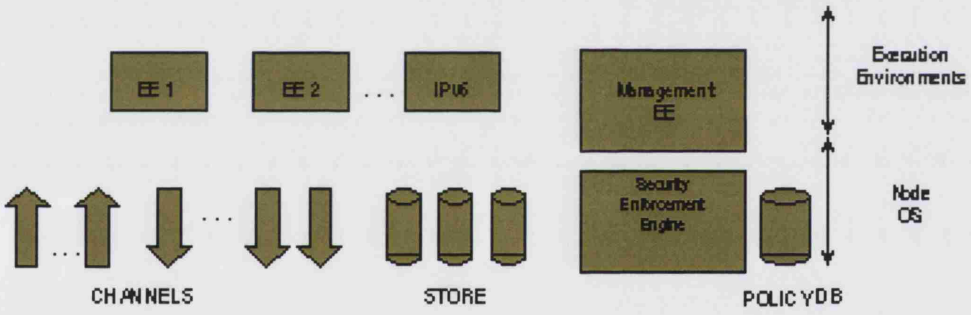


Figure 25: Basic Architecture of an Active Node

In a generic way the Active Network goals can be summarised as follows:

- To create an architecture that enables solutions for immediate and future needs to be conceived and implemented easily.
- To provide a quantifiable improvement in the number and applicability of services that can be provided in the network.
- To enable application-specified control of network resources.

5.2.1 Active platform

The DINA active platform was selected to be used for distributing the policies and monitoring the different conditions set by the policies. It is based on concepts and ideas used in the Active Bell-Labs Engine (ABLE) [134], [135] system developed in Lucent Technologies. DINA is a modular and scalable software architecture that enables one to deploy, control, and manage active services (sometimes called sessions or active sessions) over network entities such as routers, WLAN access point, media gateways, and servers that support such services in IP-based networks. In addition to the deployment control and management capabilities, DINA provides a scalable, platform independent interfaces that can be used by the active services in order to manage, control, retrieve information or perform other operations in the local node. The DINA active platform consists of an Active Engine and a Forwarding Element, namely router, WLAN access point, media gateway, etc. (see Figure 26).

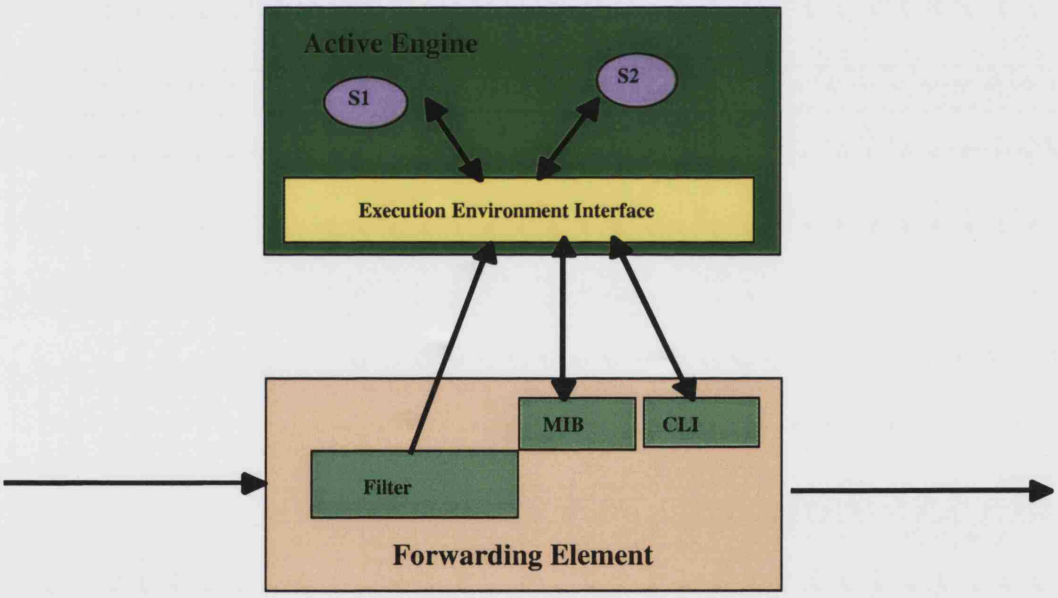


Figure 26: Active Platform Environment

The modular design of DINA allows the different components to be logically separated. In particular the Active Engine and the Forwarding Element can be either physically separated or co-located at the same machine. This kind of architecture enables DINA to support different platforms of different vendors using almost identical software component implementations.

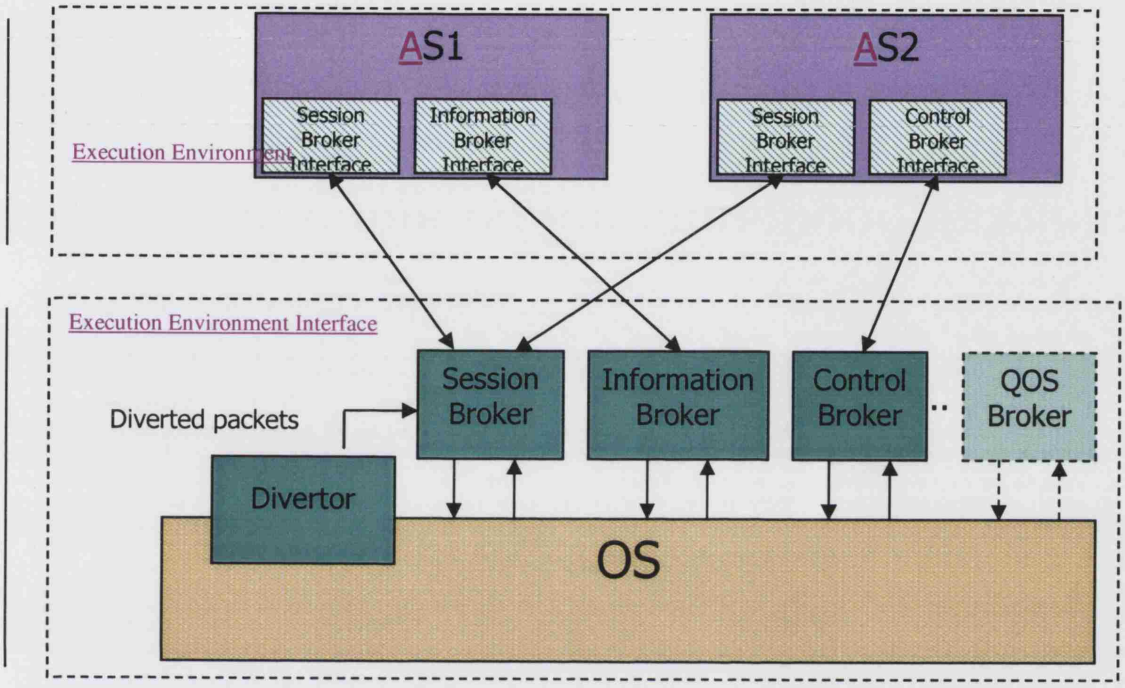


Figure 27: DINA's Block Diagrams

5.2.2 Software Modules

DINA is modular (Figure 27) software that consists of two main components, the Session Broker and the Diverter that runs in parallel on each active node. In addition to these two components, other components, which are called brokers, can run (in parallel) and give enhanced services to the active sessions using broker interfaces. The communication between DINA's modules is usually done by UDP transactions and TCP connections, although other methods can be used as well.

5.2.2.1 Diverter

DINA active packets that should be captured by active nodes in the route are identified as UDP packets with destination port. In order to capture such packets, an active node should employ a policy-based routing mechanism that enables filtering and capture

packets according to predefined rules. Using this kind of mechanism, an active node should capture such active packets and direct them to the Session Broker.

The Diverter is a simple application that receives active packets, which were captured by an active node and forwards them to the Session Broker. The Diverter may perform supplementary filtering in order to retrieve non-active packets from the network.

The implementation of the Diverter depends on the platform and the interface with the policy-based routing mechanism, in which the active node is employed.

5.2.2.2 The Session Broker

The Session Broker is the core on the active node. It receives and parses active packets, handles and manages existing services, and it distributes active packets according to requests of the services.

The Session Broker is an event driven application that receives events from different communication channels and from an ageing mechanism.

It communicates with the services via the Session Broker Interface of each service. Unlike other brokers, the Session Broker interface is essential for the execution of any service in the system.

5.2.2.3 Brokers

The brokers are modules that give active services the capability to utilise host information and resources and perform operations in the local environment. The concept of using brokers has the following advantages:

- Improving the system security and protect it against harmful services that may cause (deliberately or accidentally) problems in the active node. The usage of brokers, in addition to other security mechanisms, prevents services having direct access to sensitive information or resources in the local environment. If a service needs to

access such resources, it must use the appropriate broker that can decide, based on its policy, whether to perform the required request or to deny it.

- Enabling advanced access control, based on the services and the required resources or information. In addition to a basic binary access control that determines which services and which users can run services on an active node, each broker can determine its own policy for utilising its resources.
- Providing identical, homogeneous, platform independent, interfaces that can be used transparently by the services. Resource and information utilisation is differing from one platform to another. The usage of brokers enables services to access these resources regardless of the platform and the system that hosts the service. Moreover, once the broker is implemented for a specific platform, services can be transparently using it without any changes.
- Scalability and Stability. Each broker provides a set of operations that can be performed by the services using the interface broker API. When new operations need to be added according to the requirement of new services, it may be done by implementing a new broker that will meet these requirements. In addition, when a broker fails, it only impacts services that use this broker while other services that do not use this broker can continue their work transparently.

Each broker module consists of a broker interface and may include other components as well. For instance, a typical implementation of a broker consists of broker interface and a broker application that interact using UDP transactions or TCP connection.

5.3 Linux Routers

Linux operating system provides enhanced features and tools for networking. Netfilter is the framework inside the Linux 2.4.x kernel, which enables packet filtering, network address translation (NAT), and other packet mangling. The actual netfilter implementation is broken into two parts, the kernel portion known as netfilter (which is included in standard Linux kernel) and the userland tool that interfaces with netfilter and creates the rulesets, iptables.

Linux routing functionality and tools enable one for example to build Internet firewalls based on stateless and stateful packet filtering. For Internet sharing purposes there are NAT and masquerading available. NAT can be used for implementing transparent proxies too. In the cases in which QoS management is needed there are tc and iproute2 tools.

5.3.1 netfilter

netfilter [111] is a set of hooks inside the Linux kernel's network stack, which allows kernel modules to register call-back functions called every time a network packet traverses one of those hooks.

The main features of netfilter are (at least) stateful packet filtering (connection tracking), network address translation (NAT) functions, flexible and extensible infrastructure, and large number of additional features as patches.

IPQueue (IPQ) is a module in netfilter, which can be used to transfer IP-packets to the userland programs. It uses a Netlink socket for actual communication between kernel and userland programs. This feature is used for more complicated packet manipulation, such as altering fields of the IP header or payload.

5.3.2 iptables

The iptables module is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains. Each chain is a list of rules, which can match a set of packets. A rule consists of a specific match criteria and target. Target says what to do with matched packets, this may be a jump to a user-defined chain in the same table or drop packet [111].

There are currently three independent tables. Which tables are present at any time depends on the kernel configuration options and which modules are present. Default tables are *filter*, *nat*, and *mangle*.

Table	Description
filter	This is the default table that contains the built-in chains *INPUT* (for packets coming into the router itself), *FORWARD* (for packets being routed through the router), and *OUTPUT* (for locally-generated packets).
nat	This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: *PREROUTING* (for altering packets as soon as they come in), *OUTPUT* (for altering locally-generated packets before routing), and *POSTROUTING* (for altering packets as they are about to go out).
mangle	This table is used for specialised packet alteration. Since kernel 2.4.18, there are five built-in chains supported: PREROUTING (for packets which come to the router), *INPUT* (for packets coming into the router itself), *FORWARD* (for altering packets being routed through the box), OUTPUT (for packets which are locally-generated) and *POSTROUTING* (for altering packets as they are about to go out).

Table 2: Kernel Routing Tables

In Figure 28, the possible paths of the packets through kernel are presented. Figure 28 illustrates that the incoming network packet could be handled before the routing decision (PREROUTING), during the routing or after the routing decision (POSTROUTING). These tables can be populated with rules that would determine how a packet is to be handled.

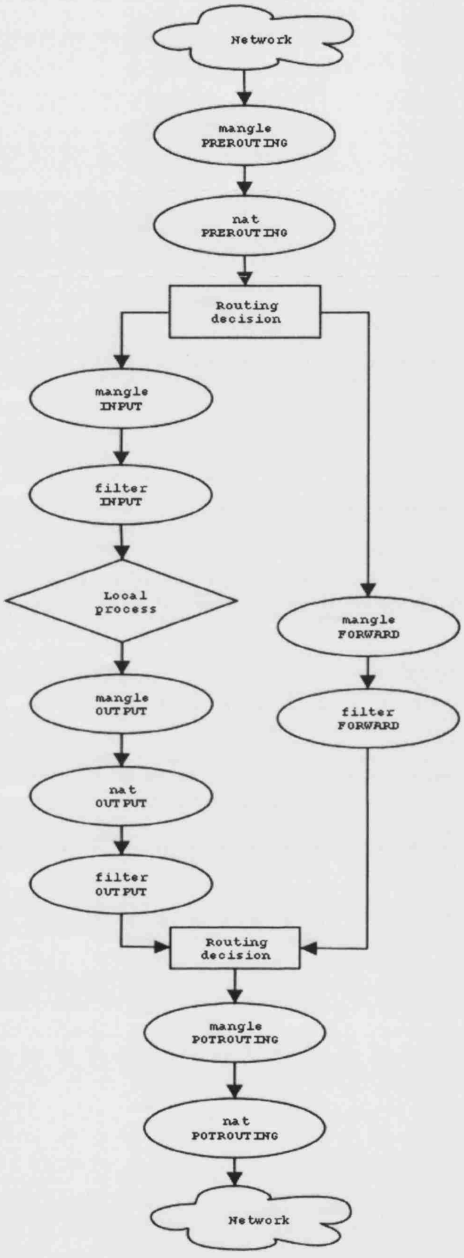


Figure 28: Packet flow through the kernel

Rules contain specific match criteria depending on what kinds of packets are needed. An example of a rule allowing forwarding packets between two networks over an interface (vtt0) would look like:

$IPT -I FORWARD -s 0.0.0.0/0 -d 10.0.0.0/16 -i eth1 -o vtt0 -j ACCEPT

$IPT -I FORWARD -s 10.0.0.0/16 -d 10.0.0.0/16 -i vtt0 -o eth1 -j ACCEPT

By properly crafting rules, one can select which rule should apply to which packets. This selection criterion can be as general or as specific as is needed. For example it may contain source/destination IP address, transport protocol, source/destination ports etc.

Netfilter has four built-in targets: ACCEPT, DROP, QUEUE and RETURN. Other targets are based on modules that load as a target. These include REJECT, LOG, MARK, MASQUERADE, MIRROR, REDIRECT and TCPMSS. Most of the targets terminate a chain. The LOG, ACCEPT, REJECT, DROPS targets do not terminate a chain, thus the chain continues to be traversed. The rest of the chain will be traversed until it hits the policy rule. The policy rule is the overall rule for the chain. If the FORWARD chain contains a policy of DROP, and no rules above match in the chain, the packet will terminate when it hits the policy rule. The policy rule can only be one of the built-in targets. Therefore, one cannot have MIRROR as a policy rule.

5.3.3 Linux Routers in this work

Linux Routers are used in this thesis to test the system implementation described in the previous chapter. Compared to the commercial router products, Linux routers are very cost efficient solutions because you do not need to buy a separate device for testing purposes. In addition, with the commercial routers one can only have restricted access to the router management. Instead, a Linux router offers access to the kernel functionalities allowing for any desired router management.

5.4 Policy-based management

The purpose of a policy system (Figure 29) is to manage and control a network as a whole, so that network operations conform to the business goals of the organisation that operates the network. Ultimately, achieving such control requires altering the behaviour of the individual entities that comprise the network.

The author views the policy-controlled network as a state machine, where policies control the policies itself (*i.e.*, meta-policies), as well as the general state of a network element at any given time. Policies can be applied using a set of policy rules. Naturally, each policy

rule consists of a set of conditions and a set of actions. It could thus be interpreted that a policy is a specification of the behaviour or actions to be taken in the managed environment; and as such, may be reactive (require an external event to trigger them) or may be proactive (*e.g.*, policies are invoked as a consequence of temporal information).

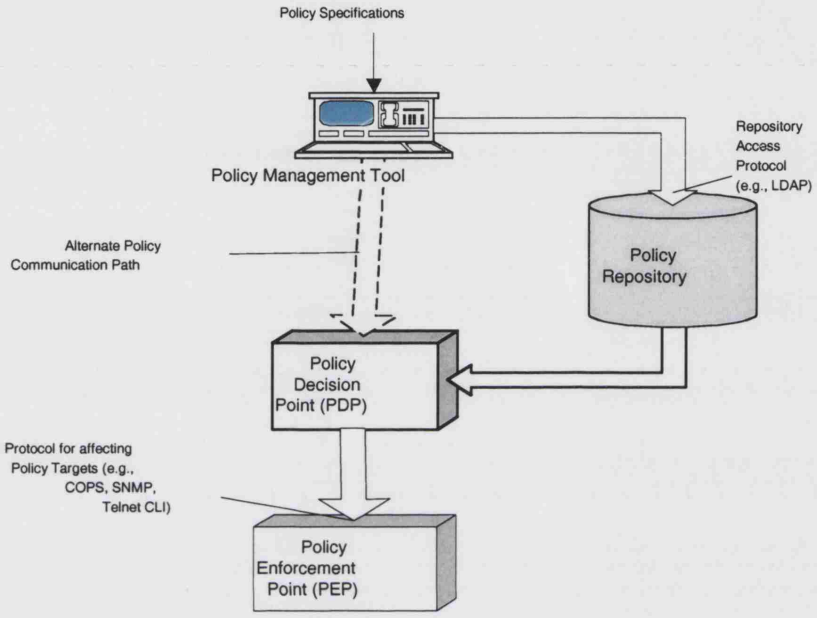


Figure 29: The IETF PBNM Reference Model

5.5 Adaptation / abstraction of network and service types

The heterogeneous nature of the physical layer of the Internet means that when information about the underlying network is needed, there is demand for an abstraction layer for the applications. In this thesis this abstraction is done by Active Networking platform called DINA. The platform includes the brokers for each of the network types used in the services/scenarios. With the help of these brokers, applications are able to manage and control underlying layer according to policies. In the next sections, the main features of the network type (WLAN) and a service type (SIP), used in the service/scenarios are described.

5.5.1 Wireless LAN

Electronic device (*e.g.*, computers and telephones) users are increasing demand for *mobility* and *flexibility*. Wireless connectivity allows a greater deal of free movement on the part of the network user and the most obvious advantage of wireless networking is mobility. Moreover, wireless networks typically have a great deal of flexibility, which can be translated into rapid deployment: Adding a user to a wireless network is a matter of configuring the infrastructure, but it does not involve running cables, punching down terminals, etc.

Apart from these advantages, there are also disadvantages inherent in the wireless network. Servers and other data centre equipment must access data, but the physical location of the server is irrelevant. As long as the servers do not move, they may be connected to wires. It also has to be said that the speed of wireless networks is constrained by the available bandwidth, which is also constrained by the size of the unlicensed spectrum bands. Radio waves can suffer from a number of propagation problems that may interrupt the radio link, such as multipath interference and shadows. Finally, security on any network is a prime concern. Transmissions on wireless networks are available to anyone within range of the transmitter with the appropriate antenna.

A wireless network consists of four major physical components, which are the following:

- ***Distribution system***: When several access points are connected to form a large coverage area, they must communicate with each other to track the movements of mobile stations. The distribution system is the logical component used to forward frames to their destination. Ethernet is the most usual distribution system used by WLAN.
- ***Access points***: The frames on the network must be converted to another type of frame for delivery to the rest of the world. Access points perform the wireless-to-wired bridging function.
- ***Wireless medium***: To move frames from station to station, the standard 802.11 uses a wireless medium.

- ***Stations*:** Networks are built to transfer data between stations, which are computing devices with wireless network interfaces.

The basic building block of an 802.11 network is the Basic Service Set (BSS), which is simply a group of stations, which communicate with each other. The communications take place within the basic service area. There are different types of basic service areas, which are described next:

- ***Independent networks***: These are called Independent BSS (IBSS). Stations in a IBSS communicate directly with each other and thus must be within direct communication range. Typically IBSSs are composed of a small number of stations. IBSSs are sometimes referred to as ad hoc BSSs or ad hoc networks.
- ***Infrastructure networks*:** Infrastructure BSS (never called IBSS) is distinguished by the use of an access point. Access points are used for all communications in infrastructure networks, including communication between mobile nodes in the same service area.
- ***Extended service areas***: BSSs can create coverage in small offices and homes, but practical reasons make it difficult to provide network coverage to larger areas. 802.11 allows wireless networks of arbitrarily large size to be created by linking BSSs into an Extended Service Set (ESS).

802.11 has been designed to be just another link layer to higher-layer protocols. The core elements present in Ethernet are present in 802.11. Stations are identified by 48-bit IEEE 802 MAC addresses. Conceptually, frames are delivered based on the MAC address. Frame delivery is unreliable because of the qualities of the radio channel. Table 3, defines the services that WLAN can offer.

Service	Station or distribution service	Description
Distribution	Distribution	Service used in frame delivery to determine destination address in infrastructure networks.
Integration	Distribution	Frame delivery to an IEEE 802 LAN outside the wireless network.
Association	Distribution	Used to establish the Access Point that serves as the gateway to a particular mobile station.
Re-association	Distribution	Used to change the Access Point that serves as the gateway to a particular mobile station.
Disassociation	Distribution	Removes the wireless station from the network.
Authentication	Station	Establishes identity prior to establishing association.
De-authentication	Station	Used to terminate authentication, and by extension, association.
Privacy	Station	Provides protection against eavesdropping.
MSDU delivery	Station	Delivers data to the recipient.

Table 3. Network Services

Station services are provided by both mobile stations and the wireless interface on access points. Stations provide frame delivery services to allow message delivery. They may use the authentication services to establish associations. Distribution system services connect access points to the distribution system. The major role of access points is to extend the services on the wired network to the wireless network.

Mobility is the major motivation for deploying an 802.11 network. Mobility can cause one of three types of transition:

- ***No transition*:** when stations do not move out of their current access point's service area, any transition is necessary.
- ***BSS transition*:** Stations continuously monitor the signal quality from all access points, which cover an extended service area. Distribution system stations do not need to be aware of a mobile station's location as long as it is within the same extended service area. BSS transitions require the co-operation of access points.
- ***ESS transition:*** This kind of transition refers to the movement from one ESS to a second distinct ESS. 802.11 does not support this type of transition, except to allow the station to associate with an access point in the second ESS once it leaves the first.

In addition to what was previously mentioned, it is necessary to introduce some facilities that are going to help us to solve some of the problems stated before. The first of these facilities is the Virtual Local Area Network (VLAN). A Local Area Network (LAN) as a broadcast domain, means that the end nodes can communicate with each other without the need of a router. Therefore, a VLAN (Virtual LAN) can be regarded as a group of devices on different physical LAN segments, which can communicate with each other as if they were all on the same physical LAN segment. In other words, a VLAN can be thought as a broadcast domain that exists within a defined set of devices. A VLAN consists of a number of end systems, either hosts or network equipment, connected by a single bridging domain.

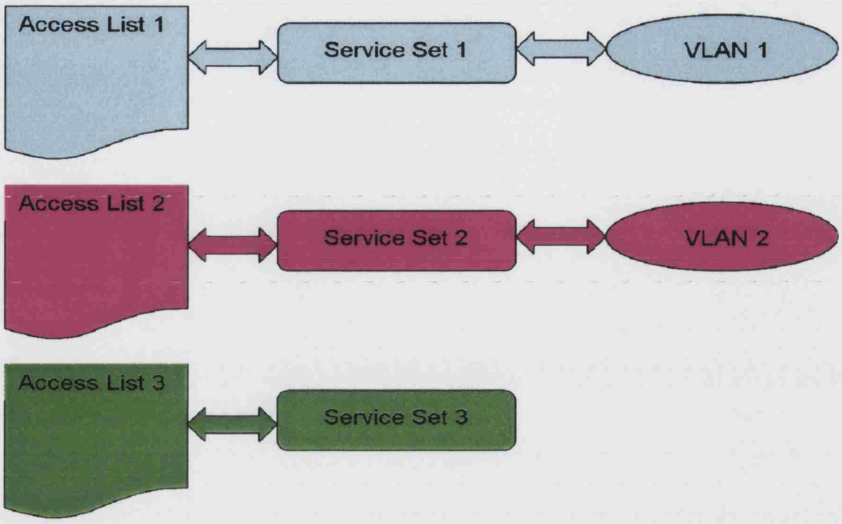


Figure 30: Schema configuration for VLANs, Service Sets and Access Lists for Access Points

When a VLAN is to be set up, it needs to be bound to a specific Service Set (SS). A Service Set can only be associated to only one VLAN. At the same time, a Service Set[38] is bound to an Access List, which will contain the MAC Addresses of all the users that are allowed to connect to this Service Set. Moreover, a given WLAN AP can also have Service Sets, which are not associated to any VLAN.

The main advantage of VLANs is that frames from the wired network with destination to clients belonging to different VLANs are transmitted by the AP to different Service Sets. Only clients associated to a particular SS can receive those packets that belong to a particular VLAN. Conversely, packets coming from clients accessing via the WLAN are 802.1Q tagged before they are forwarded onto the wired network.

5.5.2 Siptrex

The Siptrex system [115], as its name suggests, implements the functions of an IP Centrex[39]. The 'S' in Siptrex stands for Scalable. This means that the Siptrex component servers were built with system scalability in mind. Scalability is very important,

[38] Which can be viewed as a logical coverage area

[39] A Centrex has similar functionality as a PBX in classic telephony

especially if Siptrex is to be used by an Application Service Provider (ASP). It means that new businesses can subscribe to the ASP for IP Centrex services without the ASP having to purchase new hardware equipment for each new business customer. The Siptrex servers are designed to support users from multiple domains, each with different service and feature requirements. The sharing of software and hardware resources addresses one of the most important problems in the ASP world: ASPs can easily go out of business if they have to make new capital investments for every new business customer. On the other hand, a recipe for success, especially in the Communication ASP world is an IP Centrex architecture that relies on low cost components that provide a basic 'unit of functionality' that can scale as demand rises.

The Siptrex system is also useful to larger businesses that decide to converge their voice and data networks and that wish to run and manage a local installation of Siptrex, rather than subscribing to a Siptrex-enabled ASP. In both situations, the users of the Siptrex system can a) use the Siptrex User Agent as their software phone of choice, b) use a third party SIP[40] software phone, or, c) use a SIP hardware phone.

An important feature of the Siptrex system is its support for mobile users through the provision of a Siptrex User Agent Applet. As long as the mobile user has access to the World Wide Web through a standard PC with audio capabilities, the user's location is unimportant: The user can enjoy all the features and services of the Siptrex system (such being able to accept voice calls on his/her 'work' SIP URL).

5.5.2.1 Siptrex User Agent

The Siptrex User Agent (Figure 31) provides SIP soft phone functionality, as well as Siptrex specific functionality.

[40] Session Initiation Protocol

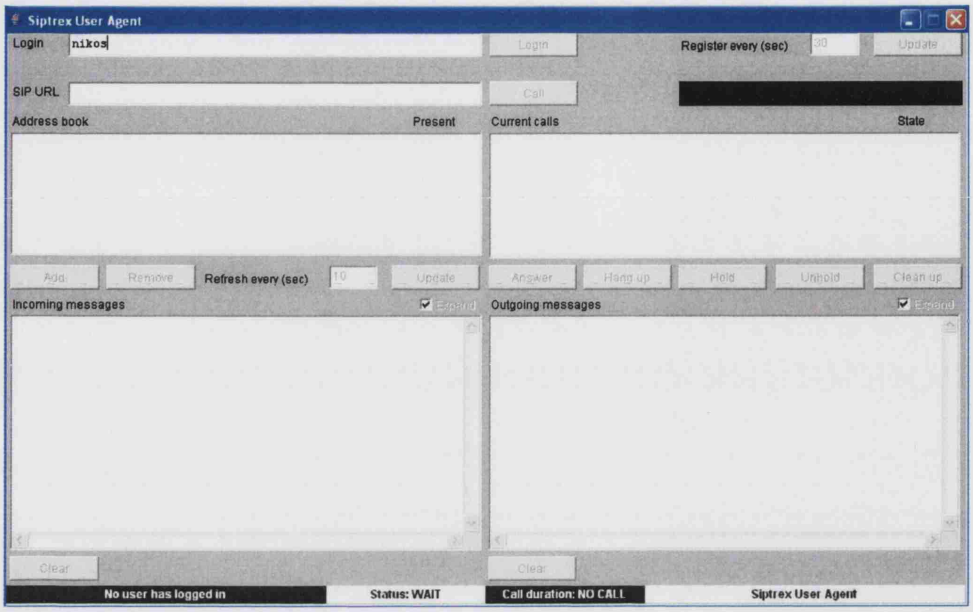


Figure 31: Siptrex User Agent

Every employee of a Siptrex-enabled organisation (that uses a private Siptrex installation or the services of Siptrex ASP) can run the Siptrex User Agent in its application or applet form.

5.5.2.2 Siptrex Feature Server

This server contains (among other things) a map of users' logins to their well-known SIP URLs. When the user wishes to login, the Feature Server is first contacted. If the user wishes to add or remove a contact on the User Agent address book, this is again done through the Feature Server and the information is stored there. When the User Agent refreshes presence information (for the people on the user's 'buddy list'), the Feature Server is contacted with a special 'REFRESH' message and it is then responsible for contacting the Siptrex Location Server and gather presence information (which is based on SIP registrations) that it then sends back to the User Agent. The communication protocols between the Siptrex Feature Server and the Siptrex User Agent run over TCP/IP and are Siptrex proprietary.

5.5.2.3 Siptrex Proxy

The second parameter the Siptrex User Agent is required to support the address of an outbound proxy (or a virtual address if the load balancing component is used). This proxy accepts all user SIP messages irrespective of SIP Request-URI. The first class of messages this proxy handles is SIP REGISTER messages that are sent periodically by the User Agent, whether they are mobile or not, in order to refresh their registrations with the Siptrex Registrar and Location Servers. The second class of messages this proxy handles is INVITE requests for users that belong to the same Siptrex domain as the caller and the proxy. The proxy must contact the Location Server and discover the current location of the user (if they are present) and forward the INVITE directly to them. A third class of messages the proxy receives are INVITE requests for users not belonging in the same Siptrex domain as the proxy and the user. These users can belong to a foreign Siptrex domain or to non-Siptrex foreign SIP domain. In either case, the proxy must discover the responsible SIP proxy (or virtual proxy in a load-balanced Siptrex based foreign domain) and forward the INVITE to it. The foreign proxy must then decide if the user is currently registered or not and respond accordingly.

5.5.2.4 Siptrex Registrar

The Siptrex Registrar is responsible for accepting SIP REGISTER messages forwarded to it by the Siptrex Proxy and communicates with the Siptrex Location Server in order to create new registrations or refresh existing ones. Registration messages are sent by the user agents. The interval between successive re-registrations is configurable by the user. Normally, a large interval is preferred, so that excessive traffic is not generated. However, this is only desirable if the User Agent runs in a "stable" environment (such as within a company during a normal day's work). For mobile users that access the system over the World Wide Web, registration every minute maybe necessary, especially because the Siptrex User Agent Applet does not implement logout functionality but relies on the REGISTER messages to refresh its soft state. If the web browser crashes or closes (if for example the PC is a public one), the fact that the Siptrex user is not logged on anymore must be propagated quickly back to the home Siptrex system (implicitly, due to

the absence of REGISTER messages), so that the Location Server can then delete the registration and so that presence information on other users' Siptrex User Agents be updated accordingly. The presence and registration information is not real-time: if a user attempts to call another user that is, in fact, not connected any more, a call timeout will eventually inform the user of this fact (The Siptrex Registrar uses the JAIN SIP API).

5.5.2.5 Siptrex Location Server

The Siptrex Location runs at the heart of a Siptrex installation and the Siptrex Proxy, the Siptrex Registar and the Siptrex Feature Server require its services. Its basic functionality is to maintain the database that maps users' SIP URLs to their current Contact SIP URLs. If a user is present in the database, the user is (probably) available and can be reached on a SIP phone. The Location Server database is purged periodically and expired registrations are deleted. Also, whenever a specific query for a SIP URL arrives, if the address has, in fact, expired, the entry is deleted and an appropriate message is returned. Expiration times are relative to the time of registration and they depend on the value of the REGISTER message Expires Header (or the expires parameter in the SIP Contact Header) as send by the Siptrex User Agent, or any other standard SIP User Agent.
The communication protocols between the Siptrex Location Server and the Siptrex Registrar, Proxy and Feature Server run over TCP/IP and are Siptrex proprietary.

5.5.2.6 Siptrex Redirect Server

This is an optional component that serves to provide basic redirection services for SIP User Agents and Proxies. It is normally a logical function of the Siptrex Proxy and it uses the Location Server to obtain redirection information. The Siptrex Redirect Server can run as a separate JAIN SIP entity.

5.5.2.7 Siptrex modifications

In order to use Siptrex platform in the services section (7.3), the following modifications to the Siptrex User Agent had to be made:

- New configuration parameter added.

A new configuration parameter had to be added to the User Agent; the IP address of the SIP Broker[41] (described in section 7.3.8.1). This parameter will be used when a subscriber tries to make a SIP call to another user.

- A Blocking call to SIP Broker

  When attempting to make a SIP call, a blocking call to the SIP broker is made requesting authorisation. Unless the SIP Broker responds no call is processed by the Siptrex. If the SIP Broker authorises the call, then the call process continues as normal, otherwise no call can be made.

- A SIP Broker Listener

  A Listener was developed on the User Agent so it can receive Information and Termination messages from the SIP Broker. This Listener, listens to a dedicated port for messages from the SIP Broker. Through this, the SIP Broker can inform the user the reasons why a call was rejected and also drop a call if the user is not authorised to use the SIP resources.

5.6 Conclusions and Summary

This chapter presented the different technologies used in order to implement and eventually test the CAPL Framework. Some of the technologies presented in this chapter will be used in the following chapter, in order to build some test services, through which the use of the CAPL Framework will be demonstrated.

The use of an Active Platform (DINA) for distributing the policies was decided. DINA is based on concepts and ideas used in the Active Bell-Labs Engine (ABLE) system developed in Lucent Technologies. DINA is a modular and scalable software architecture that enables the deployment, control, and management of active services (sometimes called sessions or active sessions) over networks entities such as routers, WLAN access point, media gateways, and servers that support such services in IP-based networks.

Furthermore the use of Linux routers and their features was also decided. Linux Routers are used in this thesis to test the system implementation described in the previous chapter.

[41] The SIP Broker component was developed in order to control the SIP environment be able to enforce the policy actions to it. It offers an interface to the Policy Based Management System.

Compared to the commercial router products, Linux routers[42] are very cost efficient solutions because you do not need to buy a separate device for testing purposes. In addition, with the commercial routers one can only have restricted access to the router management. Instead, with a Linux router, access to the kernel functionalities (from where one can manage the router as chooses) is available.

Finally, one network and one service abstraction were presented (WLAN and SIP). Both of them will be used as the basis for the service scenarios developed in chapter 7, which will be used to test the way the CAPL Framework operates.

[42] Apart from Linux routers, commercial routers can also be used. In order to do so, the router specific commands needed for configuration must be mapped to the the policy enforcement points functionality through the use of appropriate development and use of a broker (example of this can be found in chapter 7); eg CISCO operating System commands mapped to the WLAN Broker.

6 Component Testing

6.1 Introduction

This chapter describes the functional testing[43] of each of the CAPL Framework components, in order to verify their functionality. Testing the individual component functionality is very important, so that when the components are integrated, only minor fine-tuning would be required. The functional tests will be used to check the correct operation of the core components of the policy engine. These components contain the intelligence of the policy processing and are the basic components of the policy engine itself and are:

- The Policy Management Tool
- The Conflict Checker
- The Policy Storage Service
- The Policy Manager & Decision Maker

6.2 Policy Management Tool

Testing the Policy Management Tool requires the following to be tested:

- Policy Editor
- The Preferences Menu

6.2.1 Policy Editor Test

- *Test Procedure*

The PMT and the Policy Manager Interface are loaded. In order to test the Policy Editor, a policy had to be authored. Once that was done, from the File menu of the editor the option 'Insert Policy' should be selected.

[43] Similar testing methodology was followed during the WINMAN and CONTEXT projects.

- ***Test expected results***

Once the 'Insert Policy' was selected, the authored policy would be sent to the Policy Manager itself for parsing and compilation. At the Policy Manager side, once the policy was received, it would be printed out for verification purposes.

- ***Test Results***

- *Policy sending:* Success. The policy was sent to the Policy Manager Interface and printed out.

6.2.2 The Preferences Menu Test

- ***Test Procedure***

The PMT is loaded.

- ***Test expected results***

From the File menu of the editor the option 'Preferences' should selected. Once that was done, the test preferences should be entered, (PSS IP: 127.0.0.1, PSS Path: /policies, username: root, password: mysql.adm). Once the values are inserted, the 'Accept' button should be used. Once that is done, the preferences should be stored in a text file at /preferences/capl.preferences.

- ***Test Results***

CAPL Preferences Set: Success. The preferences were stored in the /preferences/capl.preferences file.

6.3 The Conflict Checker Test

This test is designed in order to check that the Conflict Checker functions as required, ie pick up potential policy conflicts and notify the policy author.

- ***Test Procedure***

The Policy Management Tool, the Policy Manager, the Policy Storage Service and the Conflict Checker are running.

- ***Test expected results***

From the PMT, the first policy is authored (appendix A4.1). This policy states that if a 'Hello' event occurs, then a blue pop up window should appear. As soon as the policy is inserted into the system, it is received by the Policy Manager, who forwards it to the Conflict Checker. Since it is the first policy inserted no conflict is detected, no message is sent to the policy author and the policy is stored in the Policy Storage Service. Then the second policy is inserted into the system. The second policy (appendix A4.2) is triggered by the same event as the first one (and has the same policy group and set ids as the first one), but has a different action associated to it. This policy states that if a 'Hello' event occurs, then a black pop up window should appear. The Conflict Checker retrieves all the stored policies from the PSS and compares them one by one with the newly authored one. Since the triggering event and the policy set and policy group are the same there is a potential conflict and the policy author is notified[44] and asked if he wants to proceed with the installation of the second policy.

- ***Test Results***

- *Potential Policy Conflict identified:* Success. The potential conflict between the two policies was identified and the policy author was notified.

6.4 Policy Engine Core Components Test Specifications

The tests of the main functionality of the policy engine (Policy Manager & Decision Maker) are grouped in three different sections. The first section contains the tests regarding the Condition Evaluation process. The second section contains the specific tests regarding the Action Enforcement process. Finally the third section contains the tests

[44] A conflict notification is sent, together with the conflicting policy.

regarding the Policy Processing Strategies that the engine can support. In order to perform these tests, a Demo Policy Enforcement Point (demoPEP) was used.

The Demo Policy Enforcement Point is a Policy Enforcement Point that permits one to manually introduce the values of the variables to be evaluated in order to check the behaviour of the policy engine. Using this, the variables monitored at each moment by the PEP are known. It allows setting a value to each of these variables and observing the reaction of the policy engine. The Demo PEP communicates with the Decision Making Component in the same way as a "real" PEP would do, but doesn't monitor the condition variables in the network; it uses the values introduced manually by the administrator.

The DemoPEP is also acting as policy enforcement point. It provides only one possible action to enforce. This action will open a little pop up window with the possibility of having different background colours set. The background colour parameter is set by the policy action.

6.4.1 Condition Evaluation Tests

In order to perform these tests, a set of test policies were loaded in the engine in order to check the behaviour of the system. Afterwards various variations over this policy will be used to perform the different tests. We will call this policy ConditionTestPolicy. This policy can be found in detail in section A4.3.

The ConditonTestPolicy doesn't pertain to any Policy Set or Policy Group. The condition of the policy is composed by 7 condition objects. 5 of them are Simple Variables (var1, var2, var3, var4, var5). One of them is an Aggregated Variable (AVar1) and one is an Event (ev1).

SimpleVariables var1, var2 and var3 are of syntax Double and its values will monitored by the demoPEP as is specified in the MonitoringComponent field. The value of these three variables is used to calculate the value of the Aggregated Variable, because, as is specified in the policy, its value is (var1+var2+var3). On the other hand the Simple Variables var4, and var5 are of syntax Double and will be also monitored by the demoPEP. Finally the Event (ev1) is composed by one Event Variable (demoEvVar1) of type String, that will monitored by the demoPEP.

The condition of the ConditionTestPolicy specifies 4 different Condition Requirements (Requirement1, Requirement2, Requirement3, Requirement4). The first Requirement (Requirement1) establishes that if the value of the Aggregated Variable (AVar1) is outside the margins specified (Requirement_Type = Out_Margins) then the Requirement is accomplished. This Requirement will be evaluated by the AVCE (Aggregated Variables Condition Evaluator) component. The margins established are (5,10). The second Requirement (Requirement2) establishes that if the value of the event variable demoEvVar1 received is equal to "Hello", then the requirement is met. This Requirement will be evaluated by the demoCE component. The third Requirement (Requirement3) establishes that if the value of the Simple Variable var4 is higher than 100 then the Requirement is met. This requirement will be evaluated by the demoCE. The last Requirement (Requirement4) establishes that if the value of the Simple Variable var5 is lower than 100 then the requirement is met. This Requirement will be evaluated by the demoCE.

The policy specifies that the Evaluation Method, to be used to evaluate if the Policy Condition is met, is the following: (Requirement1 AND Requirement2) OR (Requirement3 AND Requirement4). On the other hand the policy specifies that the evaluation of the policy has to stop after 3 successful evaluations (or three enforcements of the policy actions).

Finally regarding the actions to be enforced if the Policy Condition is met, the policy specifies one action. This action will create a blue window. The action will be enforced by the DemoPEP.

6.4.1.1 TEST1: Multi Object and Multi Requirement Condition

- ***Test Procedure***

The policy engine is instantiated and the ConditionTestPolicy is loaded into the system.

- ***Test expected results***

The policy has to be received by the Policy Manager and parse it to java classes. The Policy Manager has to send the condition information to the Decision Making Component. Then the Decision Making Component has to start the PEPs that are needed in order to evaluate the condition specified in the policy. In the tests, the PEP to be used is the demoPEP. After introducing through the demoPEP the correct values in order to make the condition true (for example: var1=9, var2=2, var3=1, ev1="Hello"), the action of the policy has to be enforced, so a blue window will be expected to open.

- ***Test results***

- *Policy parsing:* Success. The policy is parsed as expected and the information contained in the xml policy file is copied to the policy Java classes, that are used throughout the engine.
- *Policy condition registration:* Success. The policy condition information is passed to the Decision Making Component as expected. Afterwards, all the condition objects are registered in the demoPEP to be monitored and evaluated.
- *Policy condition evaluation:* Success. After the suitable values were set through the demoPEP in order to make the requirements of the policy condition true, the engine informed as expected that the policy condition becomes true after the evaluation.
- *Policy actions enforcement:* Success. Due to the result of the evaluation of the condition, the engine informed that the action of the policy is going to be enforced, and therefore, a pop-up window is opened.

6.4.1.2 TEST2: Variations over the Evaluation Method

- ***Test Procedure***

The field Evaluation Method of the ConditionTestPolicy is modified in order to check if the changes take effect and the condition is evaluated differently. For this test the following Evaluation Method is used: (Requirement1 OR Requirement3) AND (Requirement2 OR Requirement4). This means that the condition of the policy will be evaluated indefinitely. After introducing these modifications in the ConditionTestPolicy, the policy is loaded to the system.

- ***Test expected results***

After introducing through the demoPEP the correct values in order to make the condition true (for example: var1=9, var2=2, var3=1, var5=90), the action of the policy has to be enforced, so a blue window has to be opened.

- ***Test Results***

- *Policy condition registration:* Success. The policy condition information is passed to the Decision Making Component as expected. Afterwards, all the condition objects are registered in the demoPEP to be monitored and evaluated.
- *Policy condition evaluation:* Success. After the suitable values were set through the demoPEP in order to make the requirements of the policy condition true, the engine informed as expected that the policy condition becomes true after the evaluation.
- *Policy actions enforcement:* Success. Due to the result of the evaluation of the condition, the engine informed that the action of the policy is going to be enforced, and therefore, a pop-up window is opened.

6.4.2 Action Enforcement Tests

In order to perform the tests, which are focused on the action enforcement capabilities of the engine, a new set of test policies were developed. These tests will try to test different aspects related with the enforcement process.

6.4.2.1 TEST1: Specification of Action Parameters through Condition Object values

- *Test Procedure*

This test will check the correct operation of the specification of the values of the Action Parameters, using the monitored values of the Condition Objects contained in the condition of the policy. In this way, the values of some specified Condition Objects (Simple Variable, Event Variable, or Aggregated Variable) will be used as the values to be given to some specified Action Parameters. The test policy to be used for this test is described in section A4.4.

The test policy specifies one condition object of type Simple Variable (var1). The requirement to be applied over this variable is of type 'Match_Value', and the value to be matched is set to 'ANY' (this is a keyword that specifies that whatever the value of the variable is, the requirement is met). The Condition is met if the unique requirement is met (see the Evaluation Method field).

In the Action part of the policy there is one single action. This action is to open a pop-up window. This action only contains one Action Parameter. This parameter specifies the colour of the window to be opened. In this example the value of the colour of the window is not a fixed value, but will be the value of the Simple Variable var1. Again, the test policy is loaded in the policy engine.

- *Test expected results*

Once the test policy is loaded in the engine, and the condition is registered in the Decision Making Component, the demoPEP needs to be used. The Simple Variable var1 of the policy is registered in the demoPEP in order to be monitored. A value for this variable is introduced. As specified in the policy, this value will be used as the colour to be used for the pop-up window that will be opened as a result of the enforcement of the action. The available colours are red, green, yellow, pink, blue, white and black.

When any of them is introduced as the value for the Simple Variable to be monitored, the condition should be met and the action specified by the policy should be enforced. So,

again the window to be opened should have the colour specified by Simple Variable var1 through the Demo Policy Enforcement Point (demoPEP).

- ***Test results***

- *Policy condition registration:* Success. The policy condition information is passed to the Decision Making Component as expected. Afterwards, all the condition objects are registered in the demoPEP to be monitored and evaluated.
- *Policy condition evaluation:* Success. After the suitable values were set through the demoPEP in order to make the requirements of the policy condition true, the engine informed as expected that the policy condition becomes true after the evaluation.
- *Policy actions enforcement:* Success. Due to the result of the evaluation of the condition, the engine informed that the action of the policy is going to be enforced, and therefore, a pop-up window was opened. The colour of the window was the one specified through the simple variable 'var1'.

6.4.2.2 TEST2: Use of local variables

- ***Test procedure***

The local variables or policy deduced variables can be understood as high level variables defined at policy level. The values of these variables can be modified as a result of some policy action and can be involved in the evaluation of other policies. This kind of variable is the chain to allow interleaved policies. This kind of variables only exists at policy level and will be calculated, stored and managed locally by the PBMS.
We are going to use two test policies for this test. The first one will be used to test how to create and give a value to a local variable. The creation of a local variable is done using an internal Action Consumer called LocalVariableActionConsumer. So, the creation and modification of a local variable is done through a policy action as a result of a policy

enforcement. The second test policy will be used to show how this local variable, previously created and initialized by the first policy, can be used as a Condition Object in order to use its value to evaluate the condition of the second policy. This is an example of interleaved policies. The two test policies to be used can be found in appendix A4.5 and A4.6.

In the first policy, the local variable is created using the action 'setLocalVariable'. The action variables specified are the Group, the Name, the Syntax and the Value. The Group ("LVGroupExample") allows to group different local variable in different groups, just for management purposes. The Name ("Var1") specifies the name of the local variable to be created of modified if already exists. The Syntax ("String") specifies the syntax type of the variable, and the value ("blue") specifies the value to be set to the local variable.

The condition of this policy will be met if its value is "hello world" (Simple Variable "var1").

The second policy will use the local variable previously created by the first policy as a Condition Object of the condition. The value of this local variable will be used to evaluate the condition. The component responsible for evaluating the local variable is a special internal component called LocalVariableCE. The condition requirement for this second policy specifies that if the value of the local variable is "blue" then the condition is met. The action of the second policy will simply open a pop-up window.

- ***Test expected results***

The first policy condition will be made true (by setting var1="hello world"). Then by means of the enforcement of the first policy the local variable "lvar1" will be created, and will initialised to the value "blue". The next step will be to load the second policy (localVarPolicy2) into the engine. The condition of the second policy will be registered into the Decision Making Component and then sent to the LocalVariableCE in order to be evaluated. The condition requirement of the second policy will be met because the value of the local variable is the one specified in the requirement ("blue"). So, the second policy will be enforced, and a window will open.

The final step is to repeat the test changing the initial value of the local variable (i.e. something different from "blue"). Then, the second policy will not be enforced because its requirement is not true in this case.

- ***Test results***

- *Policy condition registration:* Success. The policy condition information of the two test policies is passed to the Decision Making Component as expected. Afterwards, all the condition objects are registered in the demoPEP (for the fist policy) and in the LocalVariableCE (for the second policy) to be monitored and evaluated.
- *Policy condition evaluation:* Success. After setting the value of var1 to "hello world" through the demoPEP (in order to set the condition requirements of the first policy to true) the engine informed as expected that the policy condition becomes true after the evaluation. Regarding the second test policy, the condition is evaluated successfully using the value given to the local variable.
- *Policy actions enforcement:* Success. Due to the result of the evaluation of the first policy condition, the engine informed that the action of the policy is going to be enforced, and therefore, the local variable is created and initialised with value "blue". Regarding the second policy, a pop-up window is opened as expected. For the second part of the test, since the condition is not met, nothing happens.

6.4.3 Policy Processing Strategies Tests

These tests are focused on checking the capabilities of the engine related with policy process strategies. The policies are structured hierarchically in Policy Sets, containing Policy Groups, containing single Policies. The Policies contained in Policy Groups are ordered following some sequence. This sequence is the one used by the PM to decide which Policies will be activated at each moment. The first Policy in the sequence inside the first Policy Group will be activated the first. The second policy in the sequence will be activated either at the same time or after the first Policy is enforced. If the first Policy

is labelled as Atomic, then the second Policy will be activated only if the first one has been previously enforced. If the first Policy is labelled as not Atomic, then the second Policy can be activated simultaneously with the first one. The different Policy Groups inside a Policy Set are also ordered with a sequence number. The first Policy Group in the sequence will be processed first, then the second one. The Policy Groups can be also labelled as Atomic or not. If the Policy Group is Atomic, then all the Policies contained have to be previously enforced before activate the Policies of the second Policy Group. Each Policy Set will be treated independently by the PM.

6.4.3.1 TEST: Policy Sets and Policy Groups processing

- ***Test procedure***

In this test the Policy Set and Policy Group processing will be checked. In order to do that, a set of test policies grouped into policy sets and policy groups will be used.
For the test, 1 policy set will be created. It will contain 3 policy groups, and each of these groups will contain 3 policies.
The policy set used is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
    <Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Policy_Set>
        <Policy_Set_Id>PsetTest1</Policy_Set_Id>
        <Policy_Set_Aim>new</Policy_Set_Aim>
    </Policy_Set>
</Policy>
```

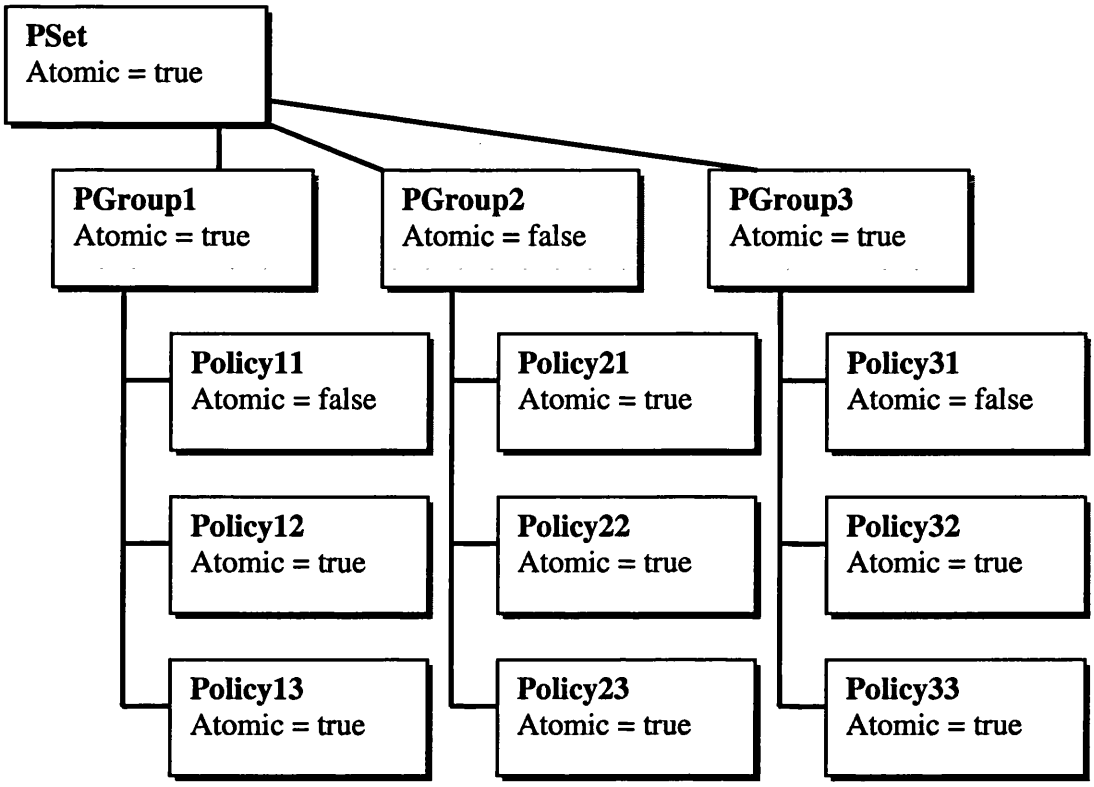
and the policy group to be used is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
    <Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Policy_Group>
        <Policy_Set_Id>PSetTest1</Policy_Set_Id>
        <Policy_Group_Id>PGroupTest11</Policy_Group_Id>
```

```
        <Policy_Group_Aim>new</Policy_Group_Aim>
        <IsAtomic>true</IsAtomic>
        <Number_Of_Policies>3</Number_Of_Policies>
        <Policy_Group_Sequence_Position>1</Policy_Group_Sequence_Position>
    </Policy_Group>
</Policy>
```

Note that the policy group has to specify the sequence number and the number of policies that it will contain. In the case of the example, the sequence number is 1, so this is the first policy to be processed of the policy set "PSetTest1". The number of policies that this group will contain is 3. On the other hand this policy group is labelled as atomic. That means that all the three policies that compose the group have to be enforced before starting to process the policies of the second group of this policy set. The policy to be used in the test can be found in section A4.7.

All the policies used are simple ones. The conditions of all the policies that will be used in this test will be met by giving the value "hello" to the simple variable that compose the unique condition object of the condition. The action in all the cases will be to open a pop-up window announcing that the action has been enforced. The key parameter that will be modified among the different policies is the IsAtomic field. The following tree summarises the policy structure that will be used for this test.



All the policies will be loaded into the policy engine in order to start the test.

- ***Test expected results***

Once all the sets, groups and policies of the test are loaded in the system the conditions of policy11 and policy12 should register to the demoPEP. This is because the policy group 1 is labelled as atomic which means that until all the policies of this group are enforced successfuly for the first time, any of the policies belonging to the policy group 2 will be registered in the demoPEP in order to be evaluated. The condition of the policy 13 is not registered because the condition of the policy12 is labelled as atomic, its condition won't be registered evaluated until the policy12 is enforced for the first time. So, if the condition of policy12 is set to true and is enforced, then the condition of the policy13 should register itself in the demoPEP to be evaluated.

When all the three policies of the policy group 1 are enforced at least once, the condition of the policy21, policy31 and the policy32 will be registered to be evaluated. The first

policy of the policy group 2 is labelled as atomic, so until this first policy of the second group is enforced, the condition of the rest of the policies (of policy group 2) won't be registered to be evaluated. Policy group 2 is labelled as not atomic, so the policies of the policy group 3 will be also processed. The first policy of the group 3 is then registered, and the second policy will be also registered because the first is labelled as not atomic. The policy 33 is not registered because the previous one (policy32) is labelled as atomic. If the policy 32 is enforced then the condition of the policy 33 must be registered to be monitored and evaluated by the demoPEP. If policy 21 is enforced then the condition of the policy 22 must be registered to be monitored and evaluated by the demoPEP. Finally if the policy 22 is enforced then the condition of the policy 23 must be registered in the demoPEP to be monitored and evaluated.

- ***Test results***

- *Policy condition registration:* Success. The policy condition information of the test policies is passed to the Decision Making Component as expected, following the expected sequence as described in the test expected results section. Afterwards, all the condition objects are registered in the demoPEP to be monitored and evaluated.
- *Policy condition evaluation:* Success. After the suitable values were set through the demoPEP in order to make the requirements of the policy condition true, the engine informed as expected that the policy condition becomes true after the evaluation.
- *Policy actions enforcement:* Success. Due to the result of the evaluation of the policy conditions, the engine informed that the action of the policy is going to be enforced, and therefore, a pop-up window is opened as expected.

6.5 Policy Storage Service

In order to be able to test the Policy Storage Service (PSS), a testing component was developed. From that component, policy database elements (policy Sets, policy Groups

and Policies) could be added, retrieved or removed. This was done by invoking methods offered by the PSS interface. In order to be able to perform these tests, an SQL database needs to be installed and configured.

6.5.1 Adding a PolicySet

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the XML file which contains the PolicySet. The PSStest will then read the policySet from the file and use the 'addPolicySet' method of the PSS interface with input parameters: policySetId and *policySetXML*. This method issues an SQL query of the form: *'INSERT INTO policySetT VALUES (policySetId, policySetXML);'*

- ***Test Results***

PolicySet addtion: Success. From the SQL Query Browser, the PolicySet can be found added into the database

6.5.2 Adding a PolicyGroup

- ***Test Procedure***

. The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the XML file which contains the PolicyGroup. The PSStest will then read the policyGroup from the file and use the 'addPolicyGroup' method of the PSS interface with input parameters: policySetId, policyGroupId and *policyGroupXML*. This method issues an SQL query of the form: *'INSERT INTO policyGroupT VALUES (policySetId, policyGroupId, policyGroupXML);'*

- ***Test Results***

PolicyGroup addtion: Success. From the SQL Query Browser, the PolicyGroup can be found added into the database

6.5.3 Adding a Policy

- ***Test Procedure***

. The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the XML file which contains the Policy. The PSStest will then read the Policy from the file and use the 'addPolicy' method of the PSS interface with input parameters: policySetId, policyGroupId, policyId, policyXML and *policyJava*. This method issues an SQL query of the form: *'INSERT INTO caplf2 VALUES (policySetId, policyGroupId, policyId, policyXML, policyJava);'*

- ***Test Results***

Policy addtion: Success. From the SQL Query Browser, the Policy can be found added into the database

6.5.4 Retrieve a PolicySet

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 2 (Retrieve PolicySet) with policySetId PolicySetId1. This was the policySet previously inserted into the database through the use of the addPolicySet. This will invoke the retrievePolicySet method of the PSS interface with input parameters: policySetid1. This method issues an SQL query of the form: *'SELECT policy FROM policySetT WHERE (policySetId=policySetId1);'*.

After this the policySet with policySetId=policySetId1 would be displayed by the PSStest.

- ***Test Results***

PolicySet retrieval: Success. The policySet with policySetId= policySetId1 is displayed by the PSS.

6.5.5 Retrieve a PolicyGroup

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 3 (Retrieve PolicyGroup) with policySetId= policySet1 and policyGroupId=PolicyGroup1. This will invoke the retrievePolicyGroup method of the PSS interface with input parameters: policySetid1 and policyGroupId1. This method issues an SQL query of the form: '*SELECT policyGroupT FROM policy WHERE (policySetId=policySetId1 AND policyGroupId=policyGroupId1);*'. After this, the policyGroup with policyGroupId= policyGroupId1 would be displayed by the PSStest.

- ***Test Results***

PolicyGroup retrieval: Success. The policyGroup with policySetId= policySetId1 and policyGroupId= policyGroupId1 is displayed by the PSS.

6.5.6 Retrieve a Policy

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 4 (Retrieve Policy) with policySetId= policySetId1, policyGroupId=PolicyGroup1 and policyId = policyId1. This will invoke the retrievePolicy method of the PSS interface with input parameters: policySetid1, policyGroupId1 and policyId1. This method issues an SQL query of the form: '*SELECT policy FROM policy WHERE (policySetId=policySetId1 AND policyGroupId=policyGroupId1 AND policyId=policyId1);*'. After this the policyGroupId1 would be displayed by the PSStest. After this, the policy with policyId1 would be displayed by the PSStest.

- ***Test Results***

Policy retrieval: Success. The policy with policySetId= policySetId1, policyGroupId= policyGroupId1 and policyId= policyId1 is displayed by the PSS

6.5.7 Remove a PolicySet

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 5 (Remove PolicySet) with policySetId= policySetId1. This will invoke the removePolicy method of the PSS interface with input parameters: policySetid1. This method issues an SQL query of the form: 'DELETE FROM policySetT WHERE (*(policySetId=policySetId1));*'. After this the policySet with policySetId=policySetId1 would be removed from the database.

- ***Test Results***

PolicySet removal: Success. The policySet with policySetId= policySetId1 was removed by the PSS. This was verified by the use of the SQL query browser.

6.5.8 Remove a PolicyGroup

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 6 (Remove PolicyGroup) with policySetId= policySetId1 and policyGroupId=policyGroup1. This will invoke the removePolicy method of the PSS interface with input parameters: policySetid1and policyGroupId1. This method issues an SQL query of the form: 'DELETE FROM policyGroupT WHERE *((policySetId=policySetId1 AND policyGroupId=policyGroupId1));'*. After this the policyGroup with policySetId=policySetId1 and PolicyGroupId= policyGroupId1 would be removed from the database.

- ***Test Results***

PolicyGroup removal: Success. The policyGroup with policySetId= policySetId1 and policyGroupId=policyGroupId1was removed by the PSS. This was verified by the use of the SQL query browser

6.5.9 Remove a Policy

- ***Test Procedure***

The PSS and the PSStest need to be running.

- ***Test expected results***

From the PSStest, select the option 7 (Remove Policy) with policySetId= policySetId1, policyGroupId=policyGroup1 and policyId=policyId1. This will invoke the removePolicy method of the PSS interface with input parameters: policySetid1, policyGroupId1 and policyId11. This method issues an SQL query of the form: 'DELETE FROM caplf2 WHERE *((policySetId=policySetId1 AND policyGroupId=policyGroupId1 AND policyId=policyId1));'*. After this the policy

with policySetId=policySetId1, PolicyGroupId=policyGroupId1 and policyId=policyId1 would be removed from the database.

- ***Test Results***

Policy removal: Success. The policy with policySetId= policySetId1, policyGroupId=policyGroupId1 and policyId=policyId1 was removed by the PSS. This was verified by the use of the SQL query browser

6.6 Conclusions

This section presented the component functional testing of all the components developed by the author. All tests were completed successfully. This section was very important, because the functionality of each of the components needed to be verified before combining everything together in order to exercise the whole system in the next chapter through the use of two services used in simple scenarios. These services will test the system in an integrated fashion.

Initially, the Policy Management Tool was tested. This included testing the Policy Editor functionality through authoring a policy and then submitting it to be installed in the system. Also, the Preferences menu of the Policy Management Tool was tested.

Then, the Conflict Checker functionality was tested in order check if potential policy conflicts can be identified. This was done through the use of two similar policies, which had different actions associated to the same triggering events.

Having tested the upper part of the CAPL Framework, its core functionality had to be tested. This involved testing the policy engine (Policy Manager & Decision Maker) in three different areas. The first area was testing the Condition Evaluation process. The second area was about testing the Action Enforcement process. Finally, the third area was about testing the Policy Processing Strategies that the engine can support. In order to perform these tests, a Demo Policy Enforcement Point (demoPEP) was used.

The Demo Policy Enforcement Point used was a Policy Enforcement Point that permits one to manually introduce the values of the variables to be evaluated in order to check the behaviour of the policy engine. Using this, the variables monitored at each moment by

the PEP are known. It allowed setting a value for each of these variables and observing the reaction of the policy engine. The Demo PEP communicates with the Decision Making Component in the same way as a "real" PEP would do, but doesn't monitor the condition variables in the network; it uses the values introduced manually by the administrator/tester. This DemoPEP was also acting as policy enforcement point. It provides only one possible action to enforce. This action would open a little pop up window with the possibility of having different background colours set. The background colour parameter was set by the policy action.

Once these tests were completed, the Policy Storage Service functionality was tested. These tests focused on introducing new policies (Policy Sets or Policy Groups) into the policy repository, retrieving them from the repository and finally removing them.

Once all these tests were completed, the overall system had to be tested. This involved testing the interaction of the upper part of the Policy Management System (CAPL Policy Manager, the Conflict Checker, the CAPL Decision Maker, the Policy Storage Service) with the specialised Policy Enforcement Points to be tested. These Policy Enforcement Points were specialised so that the required policy could be applied by them to the appropriate network elements. This new set of tests, which will be described in the next chapter (chapter 7) will allow for testing the 'vertical integration' of the system from the policy generation to the policy enforcement on the network.

7 Scenarios and Testing

7.1 Introduction

This section describes two services/scenarios devised for testing the generic context aware policy based system developed by the author. The first one is known as the FollowMe service and the other one as the Emergency Support Service. These scenarios were developed in order to exercise the CAPL framework from different perspectives using context information of different granularity.

In the FollowMe service, network context information is being used to trigger the system whereas in the Emergency Support service both application and network context are used. In order to be able to implement those services from the CAPL generic infrastructure, specialised Policy Enforcement Points were developed for each of the scenarios. These are making use of the active platform functionality described in chapter 5, in order to monitor, receive and enforce the required policy actions. The use of the active platform, offers the capability to utilise host information and resources and perform operations in the local node environment. In this way, an identical, homogeneous, platform independent environment is created that can be used transparently. It also enables the Policy Enforcement Points to access these resources regardless of the platform and the system that hosts the service.

Each of the Policy Enforcement Points provides a set of operations that can be performed using an interface API. When new operations need to be added, the implementation of a new broker within the policy enforcement points that meets these requirements could be used to easily extend the framework. Furthermore, when one of these brokers fails, it only impacts services that use this broker while other services that do not use this broker can continue their work transparently. These scenarios will also exercise the Policy Manager functionality to create and send active code (policy actions) to the Policy Enforcement Points once the conditions of the corresponding policies are met.

Before describing into detail the exercising the CAPL framework in an integrated way, the two scenarios will be described, together with their required networking set-ups as well.

Both scenarios contain three main phases, which are:

- The Bilateral Agreement phase
- The Data Collation phase
- The Servicing and Configuration phase

During the Bilateral Agreement phase (Figure 32), the mobile services operator (vod) establishes an agreement with the context services operator (CS). Furthermore, the different network providers (wireless networks: $WiFi_1$, $WiFi_2$) register with the context services operator.

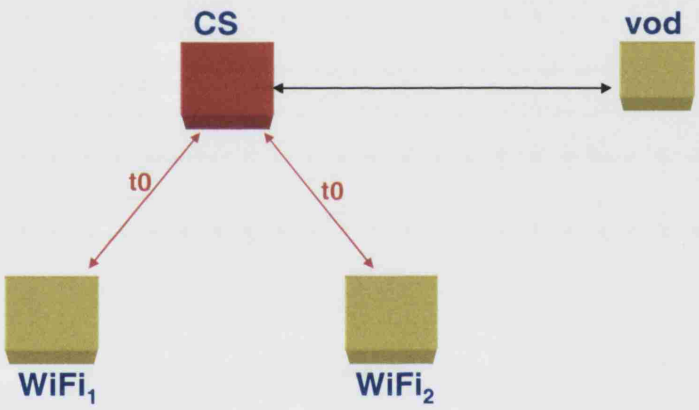


Figure 32: Phase 0 (bilateral agreements)

During the Data Collation phase (Figure 33), the main actors exchange information regarding the allowed users and possible configurations for them. This can be done over the phone or email or through the use of CORBA or SNMP.

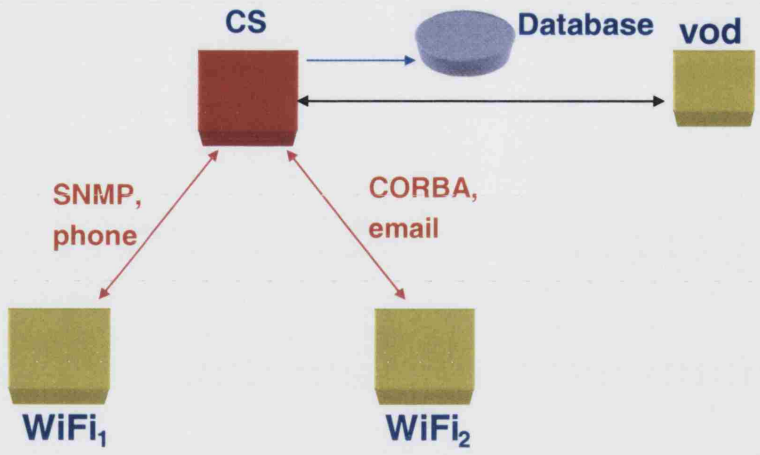


Figure 33: Phase 1 (collation)

The Servicing and Configuration phase (Figure 34) occurs when a user tries to make use of the services he has subscribed to.

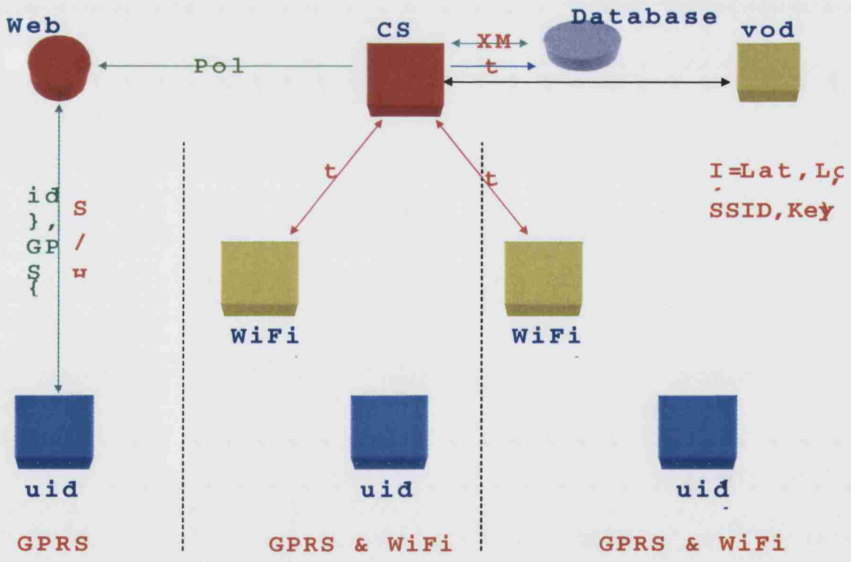


Figure 34: Phase 2 (Servicing: Authorisation, Configuration)

The user is picked up by the networking facilities and depending on his service subscription and the context aware policies related to his service (and hence the user), he can make use of the networking facilities.

7.2 The FollowMe Service

7.2.1 Scenario Description

Consider Katherine, a middle class graphic designer with 3 kids. Katherine works from home a few days a week, using her home network that is connected to the office network. On the due day of the project (she was working on in the last few weeks) Peter, her 9 year old son, complained that he does not feel well, and his situation degraded to the point that she had to take him to the local public hospital. She calls a taxi to take them to the hospital and attempts to finish her project on her way to the hospital. Halfway through to the hospital she finishes and decides to send her work to the office using the GPRS network. However, the bandwidth of the mobile network is insufficient and it would take about 50 hours for the transfer of the 1Gbyte file. In spite of this, she is not worried, as her company had recently signed up to a Context Aware Wireless Data Service (CA-WDS). This service provides an always on low bandwidth GPRS data transfer service and a very high-speed service at WiFi hotspots. Katherine is aware that there are several of these hotspots in the town, in particular at the St. Jude's Children's Hospital where she is taking Peter. She arrives with Peter at the hospital that has a WiFi hotspot. She decides to wait until she reaches the hospital to send the file. Once there, she starts the upload. The file upload will be complete in less than two hours. Katherine looks at the new transfer time estimate lets out a sigh of relief and holds Peter closer.

7.2.2 Actors description

According to the above scenario description, several actors/roles of actors are needed. However, in practice, the end user is likely to only deal with an Active Network Service Provider and an Internet Access Provider. In addition, the Active Network Service Provider may be an Internet Access Provider as well, in which case the end user deals only with a company with a role of total communication service provider, encompassing all roles exhibited towards end user. The roles related to subcontractors may exist.

End User: In the scenario the end user is Katherine. An end user uses the service provided by the Context-Aware Service Provider (CASP) transparently after she first registers herself to this service. The registration can be done for example via a register GUI or portal provided by CASP or by phone to the CASP. Furthermore, it is assumed that Katherine is connected with the CASP via the Internet and as such, the context-aware service system infrastructure and its underlying active networks based infrastructure are completely transparent to Katherine. In particular, it is assumed that Katherine does not have any knowledge on how to change the services or program the network in any way.

File Uploading Service Provider (FUSP): The File Uploading Service Provider (FUSP) offers file-uploading service to its customers. To provide this service, this actor uses virtual private network infrastructure provided by the network provider to operate the file uploading securely. It also uses the active communication infrastructure provided by the Active Network Operator in order to distribute traffic among the service nodes.

It uses the active network infrastructure via Active Network Service Provider (ANSP) in the sense that it programs active nodes so as to provide some kind of QoS or VPN.

In our scenario File Uploading Service Provider is located in the enterprise network of Katherine's company.

Context Information Provider (CIP): Context Information Provider provides the necessary context information used by Context-Aware Service Provider. CIP is also responsible for updating information of new Context-Aware services.

Context-aware Service Provider (CASP): This actor uses the Context Information Provider and Active Network Service Provider to provide transparent, scalable Context-Aware service(s) to end-users. The CASP may also operate a web server, but in general the operation of a web server may be completely outsourced. The focus of the CASP is the development of context-aware services for end-user. For this reason, minimal additional skills should be required to use the service provided by the Context Provider and Active Network Service Provider. We therefore assume that the underlying active network infrastructure is hidden as far as possible to the CASP by the ANSP.

Active Network Service Provider (ANSP): An Active Network Service Provider provides facilities for the deployment and operation of active components into the network. It provides Execution Environment facilities to Service Providers. The services provided by ANSP are based on the active network infrastructure that is provided by Active Network Operator.

Active Network Operator (ANO): This actor provides managed resources on the active network infrastructure to its customers, i.e., to Service Providers like the File Uploading Service Provider. It has complete control over the active network infrastructure and is responsible for configuring, managing, and operating its active network domain. There may be many Active Network Operators, each responsible for its own domain.

Fixed IP Network Provider (FIPNP): Fixed IP Network Provider basically provides traditional IP networks, including both IPv4 and IPv6.

Fixed IP Network Operator (FIPNO): This actor provides managed resources on the IP/Network Infrastructure and work with Active Network Operator, configuring, managing and operating the wire network resources.

Wireless Network Provider (WNP): Wireless Network Provider (WNP) provides wireless connectivity to the service providers and covers a wide span of wireless network technologies such as waveLAN, GSM, GPRS, or even 3G/4G.

Wireless Network Operator (WNO): The Wireless Network Operator (WNO) provides wireless connectivity and operability of wireless networks, and work with Active Network Operator. WLAN, GSM, GPRS or even 3G and 4G technologies can be used.

7.2.3 Relationship amongst Actors

The relations amongst the actors described in the previous section are shown in Figure 35.

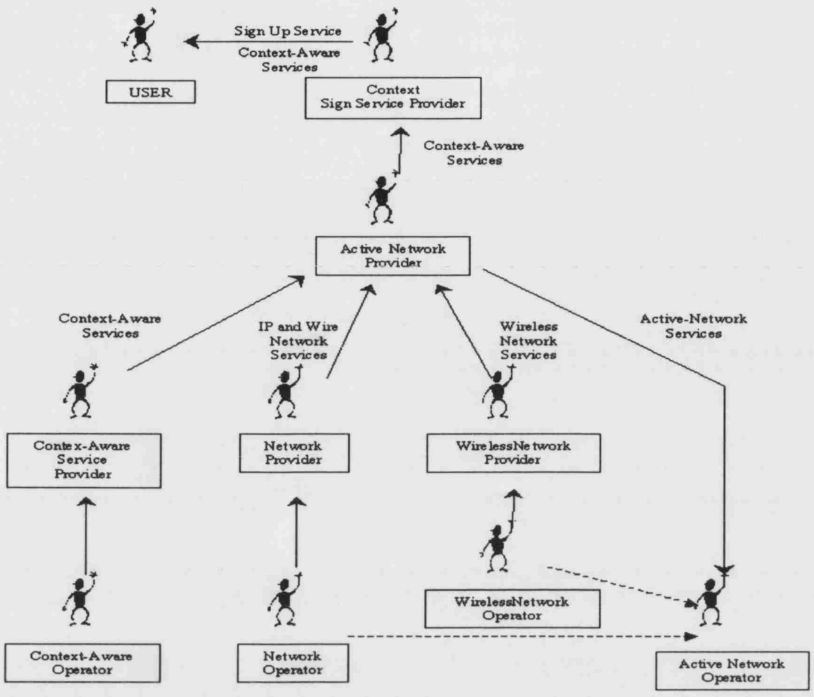


Figure 35: Interaction among Actors in the FollowMe scenario

7.2.4 Implementation Overview

The aim of this scenario is to use some context information, the policy-based management and active networking to create a flexible and transparent service to provide an always on data transfer service. This data service uses various contexts such as the user requirements and the availability of different access networks to provide the optimal data transfer service for the user. The context would be provided both by the customer and gathered by sensors and monitors. This would serve as an ideal demonstration of the use of contextual based information for the active and efficient creation of context aware services.

7.2.5 Implementation solution

Katherine's company subscribes to the FollowMe Service for its employees who wish to access data on the move. This service bundles GPRS and a Wireless LAN service and promises a usable wireless data connection almost anywhere in the country with a high speed connection at its many WLAN hotspots. These hotspots are located in many public

areas including airports, railway stations, and public buildings like hospitals, coffee shops and the central business districts. This service uses context and active networking to allow subscribers transparent and flexible handover between a GPRS and WLAN. In addition, the policy-based management system coordinates the smooth creation, deployment and execution of the very service in terms of decision-making. Value added services like VPNs to corporate servers are also provided.

7.2.6 Workflow description

Katherine's company subscribes to the FollowMe Service. Among other context, Katherine provides the MAC addresses of her WLAN cards, GPRS cards and WLAN/GPRS combo cards that she will use to access the service and other context that is deemed necessary for the creation of the service and is stored as the user and device context.

The Service Set Identifiers (SSID) of the agreed Wireless Network Providers are entered into the WLAN software. The MAC address of the GPRS/WLAN card is registered with the CASP so that it is allowed to use the service. These MAC addresses are distributed to the various access points and only these registered MAC addresses can access the hot spots points.

When Katherine requires use of the Internet and is out of range of WLAN hotspots, she can connect by using GPRS. When Katherine opens her browser, the GPRS connection is then ready to receive data. Katherine is authenticated by the credentials stored in her GPRS/WLAN combo card.

This service is defined by the policies (appendix A4.8), which are sent to the active router by the Policy Based Management System (PBMS). This is done by the Policy Manager based on the policies inserted by the administrator. Using her user context, her appropriate profile is loaded and she is ready to transmit or receive data.

If she tries to use ftp to send her files to the office, and the IP address of the ftp server is to her office or another site in which she needs VPN access, then the active router next to the GPRS network proceeds to build a secure VPN tunnel between that router and her office gateway. This is based on the policy: if Katherine's packets are destined for her office then a secure tunnel is set up for this transfer.

This is achieved by the PBMS sending active packets to the active router[45] to set up one end of an IP VPN tunnel in the GPRS domain. After this, the active packets travel from the GPRS domain all the way to the ingress router of Katherine's office to set up the other end of the tunnel. Then the customer traffic goes through this secure tunnel. To make the customer side as slim as possible (or without disturbing the user by installing any supporting software such as the DINA platform or VPN software), the secure tunnel is set up between two routers rather than between the enterprise router and customer's laptop. This is based on the assumption that there is adequate security between the laptop and the GPRS system.

Then Katherine can upload her files to the company's server. The size of the file and the speed of the connection are noted, and an estimated upload time given by her laptop.

The access point in a WiFi hotspot acts as a location sensor. When the customer is in the scope of the access point (coverage area), the active service in the access point alerts the PBMS and updates the location information. The access point authenticates her MAC address.

Using this information and other preloaded context, a new tunnel is created between the active router next to the access point and her company's VPN gateway. This process is similar to that which occurred for the VPN setup in the GPRS domain. Once the service is no longer required the VPN is taken down.

7.2.7 Network Overview

Figure 36 depicts the network that was constructed in the laboratory to support and implement the FollowMe scenario. For the FollowMe scenario, a GPRS domain and a WLAN domain and Katherine's company's network are linked via the Internet with the Context-Aware Service Provider network monitoring the overall system.

[45] A router which has the DINA platform installed

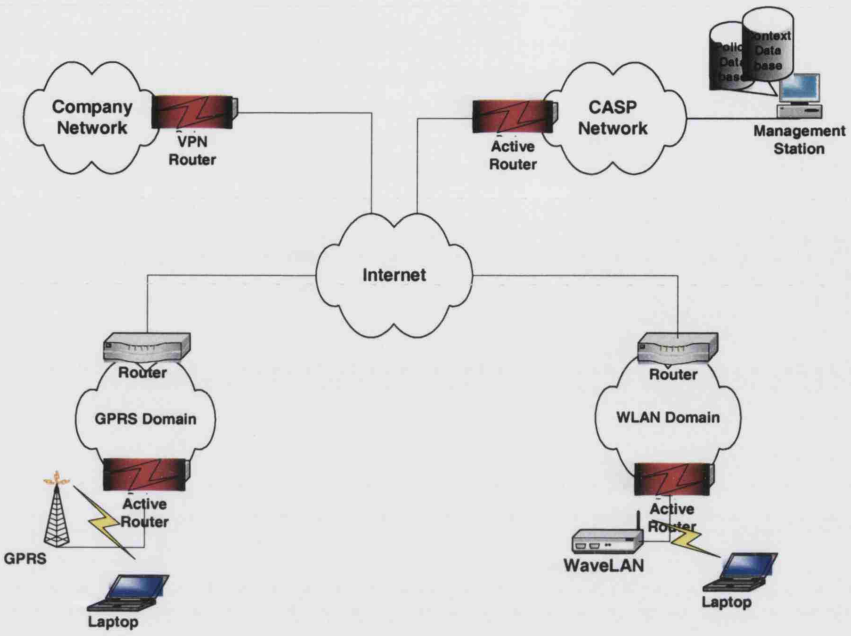


Figure 36: Network Diagram for the FollowMe Scenario

The GPRS network is a simulated GRPS network created by a WLAN, which has traffic shaping applied on the link to the Internet cloud, through the use of a Class Based Queue (CBQ) in order to simulate the GPRS environment. This was done due to the lack of access to an actual GPRS network[46]. This does not influence the scenario, since the policies are applied at the corresponding Policy Enforcement Point rather than the network itself. All the routers in Figure 36, are active Linux routers running DINA (described in section 5.2.1) on them. Each of the network access domains (GPRS and WLAN) has an instance of the WLAN Broker (described in section 7.2.8) running together with iptables (section 5.3.2) as well. The WLAN Broker[47] component acts as the Policy Enforcement Point for the FollowMe Service. Each of the network access domain routers has a Cisco 1220 access point with different SSIDs (GPRSCustNet and HospitalCustNet), connected to them offering wireless connectivity. The CAPL Framework is running at the CASP Network provider's network. Finally, the VPN router of the Company network is also an active router running DINA.

[46] The use of a GRPS card and commercial networks was considered, but was not appropriate for the scenario since none of the mobile service providers were willing to offer management interfaces to their GPRS networks.

[47] The WLAN Broker was developed by the author as a part of the DINA platform and hence is running on the active router.

7.2.8 Supporting Components

7.2.8.1 WLAN Broker specification

The WLAN Broker is the interface between the WLAN network and the system and was developed to be part of the Active Platform. This is offering the methods of accessing the WLAN functionality. A detailed view of the generalised implementation of the broker API can be found in section A5.1. This has been specialised for the CISCO 1200 Access points used in the scenario as described in the following section (7.2.8.2).

7.2.8.2 API implementation details for CISCO APs

The class WLANCISCOWrapper is the implementation of the WLAN Wrapper for CISCO Access Points.

The WLANCISCOWrapper class is composed by a set of methods which correspond to the capabilities of the AP that are offered to DINA nodes. This Wrapper communicates with the Access Point in two different ways: In the case of methods designed for configuration, it will open a Telnet session and send CISCO OS Commands; in the case of monitoring methods it mainly will use SNMP.

Table 4 contains the mapping between the API methods and the commands and services supported by the CISCO 1200 Access Point.

Method	Main commands
boolean isUserAssociated (String ClientIPAddress)	show dot11 associations all-client | include IP
public String getUserMAC (String ClientIPAddr)	show dot11 associations all-client | include Address
String getUserIP (String ClientMACAddr)	show dot11 associations all-client | include Address

String [] getAssociatedUserAddrs()	show dot11 associations all-client | include Name
int getNumberofClients()	oids[0]=".1.3.6.1.4.1.9.9.273.1.1.2.1.1.1"; this.snmpget("public",this.ROUTER,oids)[0];
String [] getAccessList()	show access-lists
int getInterfaceStatus(int ifNum)	show interfaces FastEthernet 0 | include FastEthernet

boolean setInterfaceStatus(int ifNum, boolean status)	configure terminal
	interface FastEthernet 0
	if (status = false) then no shutdown
	if (status = true) shutdown
	end

boolean isChannelActive(int ChannelNumber, int ifNum)	show controllers Dot11Radio 0 | include Channel
int getSignalQuality(String MACAddress)	show dot11 associations "+MACAddress+" | include Quality
boolean createSS(String SSID, String AutType, String AccessListID, String MaxAssoc)	configure terminal interface Dot11Radio 0 "ssid "+SSID authentication open "+ AutType +" "+ AcessListID "max-associations " +MaxAssoc end
boolean removeSS(String SSID)	configure terminal interface Dot11Radio 0 "no ssid "+SSID end
String getClientSSID (String ClientMACAddr)	show dot11 associations "+ClientMACAddr+" | include SSID
boolean establishVLAN (String VLANId, String SSID)	"configure terminal" "interface Dot11Radio 0" "ssid "+SSID "vlan "+VLANId "exit" "interface Dot11Radio 0."+VLANId "encapsulation dot1q "+VLANId

	"exit" "interface fastEthernet 0."+ VLANId "encapsulation dot1q "+VLANId "end"
boolean removeVLAN (String VLANId, String SSID)	"configure terminal" "interface dot11radio 0" "ssid "+SSID "no vlan "+VLANId "exit" "interface dot11radio 0."+VLANId "no encapsulation dot1q "+VLANId "exit" "interface fastEthernet 0."+VLANId "no encapsulation dot1q "+VLANId "exit" "end"
boolean addAccessList (String AccessListId, String permition, String MACAddr)	"configure terminal" "access-list "+ AccessListId+ " " + permition+ " " + MACAddr "end"
boolean removeAccessList(String AccessListId)	"configure terminal" "no access-list "+ AccessListId "end"
boolean setChannel(String ChannelNumber, int ifNum)	"configure terminal" "interface Dot11Radio 1" "channel "+ChannelNumber "end"
String [] snmpget(String community, String host, String[] oids)	return snmpget(2, community, 0, 0, 0, null, null, null, null, 0, host, oids);

float getLoadPerConnectionfromClient(String ClientMACAddr)	oids[0]=this.OidConstructor(ClientMACAddr,this.getClientSSID(ClientMACAddr),"273.1.3.1.1.7");
	this.snmpget("public",this.ROUTER,oids)[0]);
	Thread.sleep(10000);

	this.snmpget("public",this.ROUTER,oids)[0] The action consists in ask for the transmitted data from the client to the AP twice in 10 seconds. The difference of this data will help us to calculate the load of the link from the client to the AP.
float getLoadPerConnectionfromAP (String ClientMACAddr)	oids[0]=this.OidConstructor(ClientMACAddr,"urbion","273.1.3.1.1.9"); Thread.sleep(10000); Ans2=(this.snmpget("public",this.ROUTER,oids)[0]);
float getLoad()	In order to implement this method, we have taken profit from the previous one.
boolean setMACFilter (String FilterName, String [ ] MACAddress , String AccessListID)	"configure terminal" "access-list "+ AccessListID + " deny 0000.0000.0000 ffff.ffff.ffff" "class-map "+ FilterName "match access-group " + AccessListID "exit" "end"
boolean setVLANFilter(String FilterName, String VLANId)	"configure terminal" "class-map "+ FilterName "match vlan " + VLANId "exit" "end"
boolean removeFilter(String FilterName)	"configure terminal" "no class-map "+ FilterName
boolean setQoSPolicy(String PolicyName, String FilterName, String Priority)	"configure terminal" "policy-map "+ PolicyName "class "+ FilterName "set cos "+ Priority); "exit" "exit"
boolean removeQoSPolicy(String	"configure terminal"

PolicyName)	"no policy-map "+PolicyName
boolean bindQoS (int ifNum, String direction, String PolicyName)	"configure terminal" switch (ifNum) { case 1: "interface Dot11Radio 0" "service-policy "+ direction + " " + PolicyName "exit" "exit"
boolean unbindQoS (int ifNum, String direction, String PolicyName)	"configure terminal" switch (ifNum) { case 1: "interface Dot11Radio 0" "interface Dot11Radio 1" "no servicepolicy "+ direction + " " +PolicyName "exit"

Table 4: Mapping between API WLAN methods and CISCO access point commands

7.2.8.3 The WLANMon

Listeners are components that listen to the specific low level signals related with the invocation of a policy code. They can be either service specific or generic listeners. A given service may find it useful to configure and use any of the existing generic listeners. Listeners send an event to the PBMS if certain conditions are met. The PBMS may perform control operations on the service (e.g. start or stop its execution), based on the information received in the event and the service execution policies that have been defined. In general, a listener can be implemented to monitor the appropriate devices in different ways. Nevertheless, the active layer provides a simple and flexible platform for these kinds of jobs. Thus, the listeners are implemented as an active service that uses the different brokers to perform its monitoring operations.

The WLANMon is an example of a Listener. It monitors a WLAN access point. The goal of this listener is to identify when an authorised user enters or leaves the domain of the

access point. This piece of information is sent to the Policy Based Management System and is used to invoke or terminate an instance of the FollowMe Service.

7.2.9 Context Information Involved

Five categories of context are relevant to our scenario: user context, network context, device context, application context, and place context.

Person Entity

- User Context: This relates to information about the user of the Context Aware Service, i.e. Katherine. This included her name, company, address, preferences, profile and needs. Her job title, skills, responsibilities, agenda, and tasks are also included. The type of service that she has subscribed to along with her user certificate, passwords, and service usage summary are also relevant.

Place Entity

- Location Context: This context includes whether the user is: at home, in the office in the Hospital or on the road (coverage area), additionally includes if she is in a GPRS coverage area or in a WiFi hotspot also which particular hotspot she is using.

Object Entity

- Application Context: It encompasses the relevant information about the access to the Context Aware Services, FTPClient, FTP server characteristics.
- Network Context: This relates to the relevant information about the network obtained through the monitoring of the network, its make up and configuration. The physical aspects like the switches and routers, access points as well as bandwidth availability, capacity, latency, the endpoints of the VPN, network addresses, traffic levels, routing information, and network security are needed. The network context encompasses information about the GPRS network, WLAN network, IP network, and active network. The information about the laptop, the MAC addresses of the network cards

used to access the network along with the type of card that is being used at the moment, and the IP address assigned to it is included here

7.2.10 System in Action

There were two different WLAN domains setup (Figure 36), one simulating the GPRS domain (SSID: GPRSCustNet) and another one (SSID: HospitalCustNet), which was the WLAN hotspot the user, would try to join. These two domains were interconnected through a third party network and were routed using active routers. As described in the scenario section, Katherine migrates from her GRPS connection to a WLAN one and can still have access to her company's network (once allowed by policies).

Initially, Katherine was only registered to use a GPRS Service. This allowed her to connect to the GPRS Domain. This domain is the simulated GPRS domain with its WLAN SSID set to 'GPRSCustNet'. Her service also allowed her to connect to her company's server and have access to her files at work by using a VPN. This was done by using an IP GRE tunnel from the GPRS edge router to her company's edge router. By this way, she could transfer files to her laptop en route to the hospital (WLAN 2) to see her son. As soon as she arrives at the hospital (as soon as the laptop got within range of WLAN2), her laptop picks up the wireless network offered by the hospital but she cannot connect to that network since she has not registered to the WLAN hospital service. So, as she moved within range of the WLAN 2 nothing happened. The WLAN's Service Set Identifier[48] (SSID = HospitalCustNet) could be picked up by a WLAN monitoring utility, but she could not join that network.

Eventually, she calls her Service Provider and adds the FollowMe service to her existing GPRS account. This service allows her to use wireless networks registered with her provider as soon as she is detected. The Service Provider adds a policy[49] allowing her to be able to join WLANs. As soon as the policy was compiled and stored in the policy repository, it was checked against its conditions. Since Katherine's laptop was already

[48] An SSID is the name of a wireless local area network (WLAN). All wireless devices on a WLAN must employ the same SSID in order to communicate with each other.

[49] The policy used in this scenario is the one described in section A4.8.

within range of the access point, an event was raised (new_user_MAC_address) and the policy conditions were then met. Then appropriate active packets were fired into the network, which contained the actions set by the policies. These policies were executed sequentially; initially the service policy was activated. This policy set the FollowMe service active. This triggered the Tunnel policy, which created IP-GRE tunnels between the Hospital Active router and Katherine's workplace active router. Once these policy actions, were carried out successfully, the Firewall policy conditions were met. This policy configured the firewalls of the two aforementioned active routers in such a way so that Katherine's traffic could go through. Once this policy action, had been carried out successfully, the Routing policy conditions were met. This policy configured the routing tables of the two aforementioned active routers in such a way so that Katherine's traffic could go through. Once the final policy action was applied, the new VPN for Katherine's laptop from the hospital WLAN edge router to her company's server was set up and ready to be used. Now, the user Katherine is connected to the company's network through an IP-GRE tunnel and files stored there can be retrieved. This test successfully created a simple service, which allowed the subscribed user to connect to her company's network wherever she was and have access to her files.

7.3 The Emergency Support Service

7.3.1 Application Description

An operator is designing a network to operate in extreme emergency situations, such as terrorist attacks, extreme weather-related disaster, etc. In such cases, as demonstrated during the recent London terrorist bombings of the 7th of July 2005, the available network bandwidth should be managed in a different way, so that the Emergency services are supported appropriately. For example, in order to allow more users to access vital information, one would like to suppress non-vital traffic such as non-critical data transfer; or limit duration of non-priority calls to allow new emergency calls to be set up. In providing this context-aware service, the network, which is managed using a policy-based approach, can detect emergency situations and react accordingly to those situations.

A plane crashes in suburb area of a small town. Michelle, a frequent jogger, happened to be in a wrong place at wrong time. Unfortunately, Michelle is now seriously injured by burning jet fuel. Mark, another jogger with a cell phone, happens to arrive in the accident scene and makes call to emergency centre. Since Mark is extremely upset, he cannot tell his location. In the mean time Jake, an eye-witness of crash, is calling to a local newspaper in order to get a reward from the breaking news. There are hundreds eye-witnesses who are doing the same as Jake or are just trying to call to their families and friends in order to tell what they have just seen.

The operator has been very enlightened and has introduced appropriate policies using the Policy Management System, (which applies the policies on an Active Network) in order to offer effective emergency support and context-sensitive services. These network features of the operator's network, allow for the establishment of emergency calls, while all the other lower priority calls are blocked. Thus, Mark can place his call to 911 (Emergency Number) and his location can be estimated from the network context.

The configuration and setup of the service are automatically performed using context information such as the volume of the calls to emergency numbers, origin place of calls, etc.

7.3.2 Actor Description

The ordinary end user: Jake an eyewitness of the accident, a user non-classified as a priority person in emergency situations is trying to call at newspaper office, although the system based on policies decides not to give access to calls from that area.

The ordinary end user: Mark a jogger with cell phone, a user non-classified as a priority person in emergency situations makes an emergency call to the emergency centre. The system decides that this call can be established because, although the user isn't classified as a priority person, the number and the area where the call is originated (the point where the crashes happened) are the triggers to the system authorisation.

The privileged end user: Jane and Bob, the ambulance paramedic, are privileged end users and the system checks the line connection and when they make a call from an emergency area the context system provides all the network facilities to make calls with wide bandwidth and constant quality.

The Sign Service Provider: This actor provides the own technologies to automatically create a SLA and SLS and offers a world of context possibilities regarding users necessities as Hospitals with videophone units for example, policy and special users to cover this kind of emergency scenario.

The role of operator: The Service provider, network administrator has programmed the network with policy management system to simplify the activities and configuration process of network at the moment that an emergency happen.

The role of a AN service provider: The AN Service provider is the actor in charge of providing all the mechanisms to guarantee the Active Network functionality and operation under emergency situations inclusive. This functionality can be supported by different technologies such as common networks, Wireless Networks, Mobile Networks, etc.

7.3.3 Relationships amongst Actors

Figure 37 depicts the relationship, between the actors described in the previous sections.

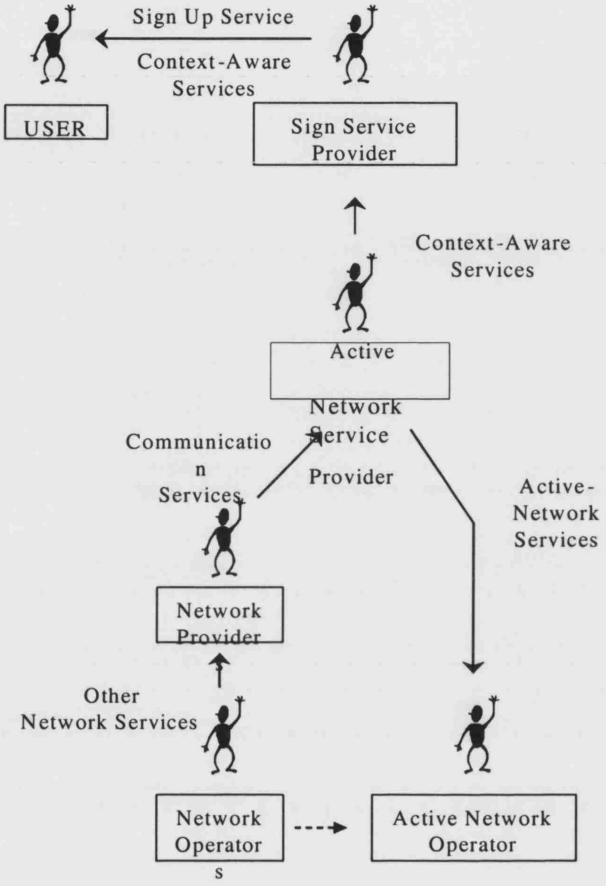


Figure 37: Interactions among Actors in Emergency Support Service Scenario

7.3.4 Use Case Description

The use case shows how actors relate to each other.

- The End User subscribes to the system through the Signs Service Provider (web page or call centre service facilities).
- The Sign Service Provider creates the relevant Service Level Agreements and Service Level Specification and purchases Active Network Services for the End User (offer all the facilities for the Context Systems).

- The AN Service provider provides all the mechanisms to guarantee the Active Network functionality and operability. This functionality can be supported by different technologies such as common networks, Wireless Networks, Mobile Networks, etc.
- The Active Network Operator has programmed the network using the policy management system to simplify the activities and configuration process of network and offer all the facilities of Active Network to the AN Service Provider.
- The Other Networks Providers/Operators offer technological resources from other technologies to satisfy the requirements of the Active Network Provider. (Common Networks, Wireless Networks, Mobile Networks, Etc.)

7.3.5 Implementation Overview

The use of context aware services is not only offered to simple home end-users. Context is important for all kind of providers, and telecommunications companies are one of the main players of the economy. They serve not only many end-users, but a lot of companies and government agencies as well. The latter are interested in ways of knowing and controlling their networks during emergencies.

The test network deployed by a communications company had been designed so that it was capable of managing itself during emergencies. All critical nodes were monitored at all times in order to detect emergency clues. The intelligence to evaluate a crisis fact was located at the active network layer. If an emergency occured, the active node would be capable of responding with some actions in order to provide support to the emergency response teams (police, doctors, fire department, etc). A central system was always updated with network status. The intelligence was modelled with policies and implemented with the Policy Based Management System.

7.3.6 Workflow description

- The active network was programmed to monitor the main communication nodes. Periodically their status was checked so that everything is normal.

- When the accident happened, many good citizens started calling the 911 emergency number. The communications network connected them with normal characteristics about duration, success ratio and quality.
- The active node in charge of the nodes that were close to the accident place made its regular checking, but now it found it was not in a normal state. A high number of calls to 911 in just one period triggered the policy about emergency events.
- The policy triggered (appendix A4.9), stated that the central system should be alarmed and some parameters of the communication node needed to be changed to support new demands. The parameters were technical ones (duration, quality) and end-user ones (if some user can/can not use the service). From that moment on, the calls were filtered according to caller and callee's profiles.

7.3.7 Network Overview

Figure 38 describes the network that was constructed in the laboratory to support the Emergency Support Service scenario.

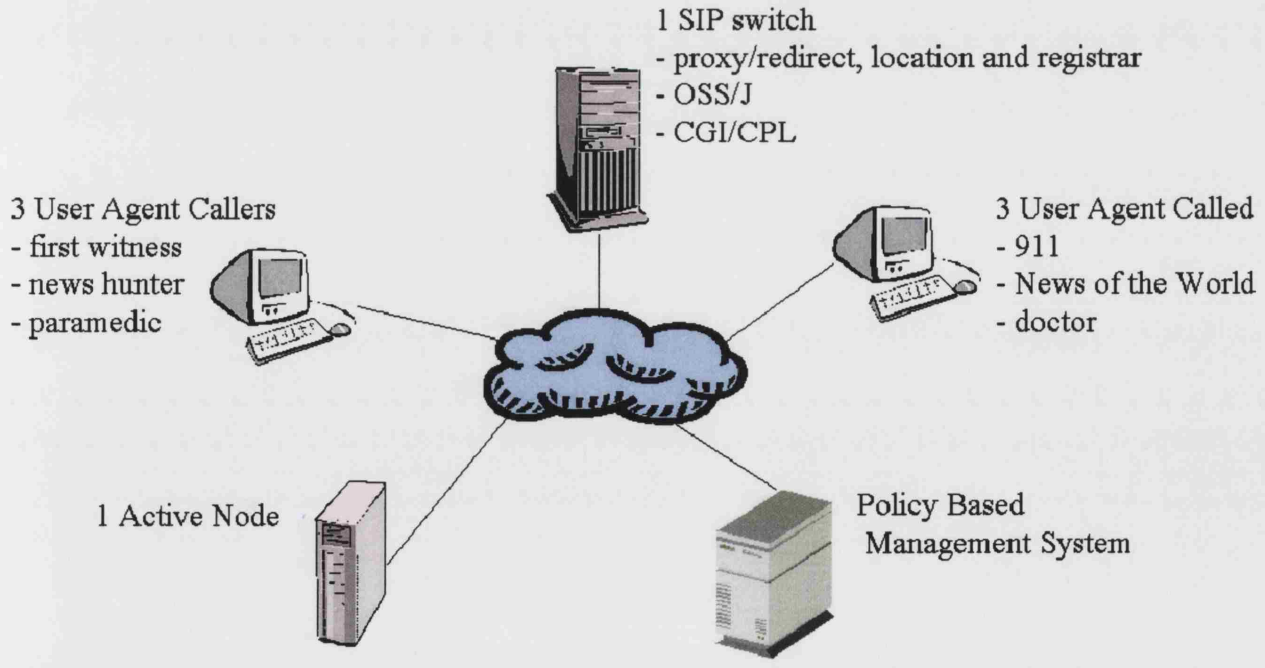


Figure 38: Emergency Support Service Network Overview

The Emergency Support Service scenario considers as the network core of the scenario, a SIP switch[50] supported by the Active networks facilities to provide the information necessary to provide efficiently context-aware services. The SIP switch offered an API for monitoring tasks and allows CGI/CPL programming. The Active Node was monitoring the SIP environment and controlled it through the use of policies. The Policy Based Management System was informed by an Active Node monitor to allow for specialised components to take some wide-area actions under the scope of the policies (Condition-Action). The test bed consisted of two Windows XP PCs running two modified siptrex agents (5.5.2), one Linux PC running the siptrex switch, one active Linux router running the SIP Broker and finally one more PC running the CAPL Framework.

7.3.8 Supporting Components

7.3.8.1 SIP Broker

The SIP Broker interfaces between the Active Application layer and the SIP software that is used to implement the Emergency Service Support scenario. The broker provides the Active Services the capability to control and manage the SIP entities such as proxy servers, and user agents. The SIP Broker provides the Policy Management System the capability to obtain information from the SIP softswitch and to control aspects of call control during a crisis situation.

It inserts a hook into SIP softswitch in order to delegate call admission to the Policy Management System. It also enables the provision of information on the SIP sessions that are taking place in the system.

The SIP broker has three primary functionalities:

- The first is to provide information on the SIP softswitch and the SIP users. The SIP softswitch is the software that controls call admission, control and signalling using the Session Initiation Protocol (SIP). The second function is to terminate

[50] This is the Siptrex platform

calls and the third is to delegate call admission to the Policy Management System. To satisfy the first functionality, it contains the methods; userStatus, getUserIP, maxSessions, totalSessions, getProxyServers, sessionStateInfo. These are all informational methods as they obtain information on the state of the system or the SIP user. Hence, the SIP broker provides an interface to provide information to the Policy Management System.

- The second functionality is the ability to terminate sessions (calls). This is done either on a call by call basis using the method terminateSession when passed the sessionID of the call and the caller and callee SIP address. This method provides the Policy Management System, the ability to terminate specific calls during a crisis situation. All the calls that originate from a specific domain (and thus handled by a specific SIP proxy server) can be terminated using the terminateAllSessions method. It takes the SIPProxyID as a parameter and provides the Policy Management System's the ability to terminate all sessions in a domain during a crisis situation. This can be useful if the network is overwhelmed by traffic from neighbouring domains during a crisis. It might be useful to terminate all calls in that domain to allow the domain(s) that have the crisis more network resources.
- The third functionality is the primary function of the SIP broker; the ability to delegate to the Policy Management System the session admission capabilities during a crisis situation. For example, calls to and from the emergency services are allowed but not to and from ordinary users. The design allows for better context information utilisation by the Policy Management System, but not by brokers, as the main goal of the work is to provide context-aware policy based management.

7.3.9 Context Information Involved

The context information relevant to our scenario as first approach is described as follows: user context, network context, device context, application context, and place context that can be closed into the entity model categories:

Person Entity: (Network End User Profile)

- User identification: Phone number, IP address, ...
- Is the user privileged?
- User Privileges (i.e. call duration; minimum BW or Quality)

Place Entity: (User Physical location)

- Physical location of network users. (i.e. to detect the emergency zone)

Object Entity: (Network Device Configuration)

- Information about the network elements configuration (GPRS, WLAN, IP...)
- Information about the devices connected to the network (laptops, mobile phones, videophones...)

Task Entity: (Call / Connection Profile)

- Calling Phone Number, Caller's Phone Number, Call duration, ..
- User's IP address, Bandwidth, Quality, ..

7.3.10 System in Action

As mentioned in section 7.3.7, the scenario/testbed consisted of a caller user pc, a callee user pc, an active node, the policy based management system node and the siptrex switch node. It is assumed that the operator, through the use of the Policy Based Management System, had installed all the relevant policies (appendix A4.9) to the system.

As soon as the accident happened and Mark (the first witness) found Michelle injured, he logged on to his SIP client and tried to place a 911 call to ask for help. As soon he attempted to call, an authorisation request was sent from his SIP client to the SIP Broker.

Since there was no emergency situation identified, the call went through. At the same time, other witnesses (other users) tried to call 911 as well. As more people contacted 911, this triggered the EmergencySupport policy (IF (911Calls >3) THEN Emergency), which set the Emergency Support service on. This in turn triggered the sipAccess policy (IF emergencyOn THEN accept EmergencyCallers policy), which only allowed authorised users to place calls. The Policy Management System then sent an active packet containing the new configuration policy for the SIP Broker enforcing the required actions.

In the mean time Jake (non privileged user), an eyewitness of the crash, attempted to call a local newspaper in order to get a reward from the breaking news. As soon as he attempted to make the call an authorisation request was again sent from his SIP client to the SIP Broker. Since an emergency situation was identified and Jake was not a privileged user, the call was blocked. Finally, the paramedic arrived at the scene of the accident and attempted to call a doctor for guidance. Both the doctor and the paramedic belonged to the privileged users group and the call was authorised to go through and was set up successfully. This test successfully created a simple service which, during an emergency could block, all but the privileged users, from making use of a SIP telephony service.

7.4 Conclusions

This section provided the testing of the CAPL Framework as a whole. The system was exercised in two different scenarios. The whole process was described using a step-by-step approach from the scenario descriptions to the network overview and finally a step-by-step view of the scenario running.

In the FollowMe service the user information and the network information were used in order to provide the user with seamless connectivity to her company's enterprise network. This was achieved by the use of context aware policies. IP-GRE tunnels were established between the two sites, firewalls were reconfigured and routing tables modified through the use of policies.

In the Emergency Support service, a set of users were attempting to make 911 calls by making use of the SIP infrastructure available in the network. Initially all calls were allowed to go through. But when the system identified a potential emergency, it reconfigured itself according to the Emergency Support Service policies, which allowed only authorised users to make use of the SIP environment.

In both services, service specific Policy Enforcement Points were developed in order to cater for the application of the different services/policies. These components offered an interface through which the policy condition variables could be monitored and the policy actions enforced. The implementation of these components made use of the active networking infrastructure offered by the use of the DINA active platform.

The CAPL Framework was exercised successfully in both scenarios and the authored policies were applied when appropriate.

8 Discussion, Conclusions & Future Extensions

8.1 Introduction

This chapter discusses and draws conclusions from the work carried out by the author. Furthermore, possible extensions to the work carried out have been identified and discussed in the future extensions section of this chapter.

8.2 Discussion

The objectives of this thesis were to define what context and context-awareness mean and to analyse the network context-awareness in networks through the use of policies. This was done through the development of a Context Policy Based Management Framework (Context Aware Policy Language and Policy Based Management Architecture) which was eventually successfully exercised by implementing two context aware services through the use of context aware policies.

In order to understand the problem space, a brief look at different approaches in network and service management were examined with emphasis placed on the Policy Based Network Management paradigm, its benefits and architecture. Policy Based Management has been a relatively new field for the Internet community, but has already been promoted by several network equipment vendors. The goal of Policy Based Management is to provide facilities, which allow the control of multiple types of devices that must work in concert to provide a desired service. Examples of devices, which can be "PBM-enabled", are hosts (clients and servers), routers, switches, firewalls, bandwidth brokers, sublet bandwidth managers and network access servers. Examples of services controlled by PBM can be Quality of Services (QoS) reservations, Virtual Private Networks (VPN), etc.

At the same time, as computer and networks become more pervasive (therefore computing is getting more pervasive), the nature of services should change accordingly. Services must become more flexible in order to respond to highly dynamic computing environments, and become more autonomous to satisfy the growing and changing requirements of users. That is, services must become more context-aware, or context-

aware services need to be provided. The author has identified what context and context awareness are, and has synthesised a novel categorisation of context awareness, from which the concept of Network context was identified as being of great importance. This was due to the fact that not much research had been invested in that area and also the fact that the network context can influence any application running on top of a context aware network.

Having identified the problem space, the next step was to describe the influences the author had in developing the required system so that the reader can better understand the proposed approach. These came, through the author's participation in two IST Research and Development projects, WINMAN and CONTEXT. The author's role in both of them was to design and implement the policy management systems controlling the rest of the functionality developed by the consortia. Through them, the author developed his design and programming skills, which were used in the development of the work described in this thesis.

The work carried out by the author and is described in the main part of the thesis, was the development of the 'Context Aware Policy Management Framework', in order to manage Context Aware Services and Networks through the use of policies. This work contains several novel attributes. In order to be able to represent context information, the author based on the new context awareness categorisation (previously defined) combined with an extended[51] version of Dey's definition of context, designed and implemented a novel object oriented context information model. Based on the novel Context Information Model developed, the author designed and implemented a novel Policy Definition Language (CAPL) for expressing context aware policies (which would include context information). This language is designed based on the IETF PCIM (Policy Core Information Model) and its extension, directions which are followed by most of the work carried out in the policy-based management field. This was done by modelling context information by using the *ContextVariable*, which inherits the abstract class PolicyVariable available in the PCIM. CAPL was developed in XML due to its flexibility and CAPL policies are written in this format as well.

[51] This extension was carried out by the author.

Furthermore, a novel Generic Policy Based Management Architecture capable of understanding and interpreting the policies expressed in CAPL was designed and implemented by the author. This allows for the authoring and application of context aware policies on a network. This PBM architecture contained a Policy Management Tool (PMT) for authoring and managing the policies, a Policy Manager (CAPLPM) for handling those policies, a Decision Maker (CAPLDM) component for making decisions, a Policy Storage Service (PSS) for storing the policies and finally some service-specific Policy Enforcement Points in order to be able to apply the policies authored.

Before implementing the system, possible enabling technologies were examined. These include all the tools and devices used in order for the CAPL Framework to be developed; from Linux Routers and their special functionalities; the use of an Active Platform (DINA); the use of WLANs to a SIP implementation (Siptrex). All of the discussed technologies were used in one way or another; either as a component of the main policy management system or as a supporting component used to monitor or apply the policies to the network.

Once the system was developed, each of the components was individually tested in order to verify their correct functionality. This was done through the use of different tests for the components. Each of the components was thought of as a state machine and therefore their different states needed to be verified.

Eventually, once the author was satisfied with the components developed, two services were designed in order to exercise the previously designed framework. The two services developed were the FollowMe Service, which allowed the subscribed user connect to his company's network from wherever he was, and the Emergency Support Service, which during an emergency could block, all but the privileged users, from making use of a SIP telephony service. From this part of the work, another in novel component could be identified. This was the fact that the monitoring of the policy conditions and the distribution of the policy actions, to the policy enforcement points was carried out by service specific policy enforcement points through the use of an Active Networking technology.

This work is attempting a step towards Autonomic Management. Autonomic management refers to the adoption of autonomously derived actions for a wide array of

management tasks. An important corollary is that the management function is itself an autonomic service, and is therefore subject to the same requirements as other autonomic services. Autonomy means self-governing; autonomic is a synonym, but has biologic connotations. The author envisions systems built from first principles that can manage themselves. The concept of self-knowledge enables the system to know its own capabilities and constraints. Policies capture goals, which define desired behaviour. Knowledge about the environment of the system, especially users of the system, enable the system to determine whether something has changed and, if so, what to do about the said change.

Self-management takes many forms. Some fundamental building blocks include:

- Self-configuration – policies direct changes to the configuration of a component or system to maintain desired functionality, or to provide new functionality
- Self-healing – system can detect, diagnose, and repair hardware, software, and firmware problems
- Self-protection, self-securing – system can automatically defend against attacks, as well as detect and stop cascading internal failures.
- Self-optimisation – system and components continually seek ways to improve their behaviour and become more efficient
- Self-learning – system learns from past events and morphs active policies to improve behaviour

Some of these combine into more powerful functions. For example, self-maintenance can be viewed as the combination of all of the above. The work described in this thesis is an attempt to move a step closer to self-configuration.

This is done through the use of context data at network-level granularity, known as Network Context. This can eventually be used to add value to a range of customer services. This would require adapting additional parameters of user- and network-centric information when provisioning for end-to-end services. The goal would be to achieve better personalisation of the user experience as per service type.

To meet these challenges there was a need to adopt an enabling technology that would facilitate rapid and efficient prototyping of service (through a set of policies), and for this

purpose, the author selected the active networking paradigm as a mechanism for customising end user traffic flow in between network edges.

In brief, the key challenges tackled by this work revolve around major issues in automation of service provisioning cycle through innovative modelling and policy-based operation and efficient context information management through active networks.

8.3 Conclusions

Although there have been many context-aware systems and applications tested over the last decade, most of them are still prototypes and only available in the research labs and the academia. One of the main drawbacks lies in the complexity of capturing, representing and processing the contextual data. The author feels that the pervasiveness of context-aware system will only be appreciated if the context can be interpreted properly.

In personal communications, people are using implicit information to increase the amount of delivered information. This implicit information is related to the context affecting the individuals or the subject they are dealing with. This is a feature that makes people to react appropriately at concepts involved in a conversation, which is not said explicitly by any of the interlocutors. Currently computers are not able to take and process all advantages that the context information has in humans' dialogues.

Clearly, it would be desirable to allow applications hosted in computers to interact with human beings making use of at least part of the information laying in the surrounding context. Due to the nature of the communication process the first step is that users and devices make explicit all the information relevant to a given situation. Nevertheless this is not easy; the problems start when most users do not know which information is potentially relevant and, what information to announce or broadcast.

From this work, the author demonstrated: how context information can be used between two different domains, *i.e.*, WiFi and simulated GPRS (in the FollowMe scenario); and how context information ensures that bandwidth allocations are realigned according to the level of urgency for the different usages (in the Emergency Support scenario). This

was carried out by the development of a context aware policy language and policy based management infrastructure capable of handling the language and enforcing it.

8.4 Future Extensions & challenges

This framework can be further extended in different directions; either towards a more research-oriented direction or towards a more implementation-oriented one. The first one would address problems of a more research nature, whereas the later one would address issues relating more to the roll out of the framework as a real system. Both of the aforementioned parameters will impose new requirements on the design and implementation of the system.

Potential research-oriented cases could involve:

- **Policy Conflict Resolution**

  Policy Conflict Resolution is a huge research topic in its own right. Its aim is to automate the identification and resolution of conflicts between policies. The identification and resolution of policy conflicts is a critical task, especially as the number of policies entered into the system increases. The inability to resolve conflicts could lead to the system behaving in an unpredictable way. Policy Conflict Resolution could be achieved by using complex algorithms. An approach similar to the one proposed by Ian Marshall [138], could prove to be useful to tackle this kind of challenge.

- **Context Information Model extension**

  The context information model developed could be extended to include different types of context information. This would allow CAPL to express a wider range of context aware policies since, as previously discussed in chapter 2, context could be any information that could be used to characterise an entity. A side effect of this extension would be the extension of the policy enforcement point functionality, in order to be able to monitor (new kinds of conditions) and enforce new actions.

Potential implementation-oriented cases could be:

- **Policy Management Tool Usability**

  The Policy Management Tool developed by the author only caters for basic functionality/usability since the PMT usability was not the main scope of the author. In order to be able to implement the framework in a real system, the PMT needs to be significantly beautified. Drag and drop components could make the authoring of policies much easier than filling in XML policy templates. Furthermore, a graphical representation of the policies and their hierarchy, through the use of a hyperbolic tree function, could also help the policy author to have a better visual understanding of how the policies are linked together. This extension could then be tested by potential users in order to quantify its usability.

- **Network-specific adaptation**

  The CAPL Framework as it is, can only cater for controlling WLANs (through the use of the WLAN Broker) and SIP Centrexs (through the use of the SIP Broker). In order to implement the framework in a real system, drivers (brokers) for the different network infrastructures need to be implemented.

- **Scalability**

  When rolling out a new framework, scalability is a big issue for system operators. In the CAPL Framework's case, its twofold; scalability problems due to policy conflicts and scalability problems due to the size of service activation requests in the system. In the first case a policy conflict resolution engine, as discussed previously, could solve the scalability issues. In the second case, bottlenecks could appear in two different places in the system; at the PEP level or at the Decision Maker level. At PEP level, the addition of a PEP for each element (overprovisioning) could solve the CPU overload problem of multiple requests arriving at a PEP. At the Decision Maker level, bottlenecks could arise if decisions need to be made based on conditions monitored by different PEPs. This

problem could be solved by increasing the CPU power of the decision making process (and hence the Decision Maker itself) by making it more distributed[52].

In addition to the aforementioned cases, various business cases that the framework could be called to support could dictate further extensions, such as when:

- **The authoring of services is left entirely to the end-user.** In this case the system must be able to impose usability, availability and stability of these per customer services by setting the appropriate restriction rules and validation checks to the authoring process. Also it should be able to ensure their management, operational integrity and performance as well as network and privacy security protection by appropriate, for such a vast number of diverse services, testing and monitoring processes. Furthermore, service templates (and hence policy templates) could be used in such a way so that the users would fill in the service templates and create the service themselves.
- **Service creation is performed by a Virtual Service Provider dependent on the capabilities of the infrastructure of the actual Service Provider.** In this case the interfaces between these two different domains must be well defined to ensure the seamless creation, management and operation of the services. In addition extra processes must be employed to facilitate the other business interactions between these domains concerning accounting, billing, customer care etc.
- **Service creation is performed by a Virtual Service Provider independent of a specific Service Provider, but able to purchase capabilities on the fly from various Service Providers.** In this case the system must provide for the means to interconnect the infrastructure of several service providers in the service management plane (PBMS) as well as in the service execution plane (AAL). For example permit the execution of services stored in domain 1 to a node belonging to domain 2, on an event received from domain 3 and ensure the retrieval of context info existing in domain 4. This, as one can easily understand, is a very demanding project that involves several issues ranging from security to scalability

[52] One way of doing this could be through the use of grid computing

to the necessary interconnection contracts and mechanisms. No matter the complexity, the author believes that it is a feasible extension and can lead to an internet-wide spreading of context aware service provisioning.

9 References

[1] Computerworld site, "Carriers put bandwidth before service", 2nd February 2005 http://www.computerworld.com.au/index.php/id;1532821864;relcomp;1

[2] Parallel Global site, "Network Diagnostics identify VSAT service provider issues for ABC", Cranfield Innovation Centre, April 2004, http://www.parallel.ltd.uk/documents/ABC_vsat_network_diagnostics.pdf

[3] WINMAN site,www.telecom.ntua.gr/winman

[4] G. Lehr, H. Dassow, P. Zeffler A. Gladisch, N. Hanik, W. Mader, S. Tomic, G. Zou, "Management of All-Optical WDM Networks"- IEEE/IFIP 1998 Network Operations and Management Symp., 1998, New Orleans

[5] Draft ITU-T Recommendation G.872 (ex G.otn) "Architecture of optical transport networks" Geneva (1998)

[6] G. Lehr, R. Braun, H. Dassow, G. Carls, U. Hartmer, A. Gladisch, H. Schmid : "WDM Network Management: Experiences gained in a European Field Trial" - IEEE/IFIP 1999 Integrated Network Management Symp.; proceedings, 485-498 pp, 24-28 May 1999, Boston

[7] ITU website, www.itu.int

[8] ODP-RM framework,' ISO/IEC IS 10746/ ITU-T X.900, http://enterprise.shl.com/RMODP/ default.html

[9] M. Hall, J Pavon, "How to use Management Architecture and resource management), TINA-C, December 1993, www.tinac.com

[10] M. Hall, G. Nilson, P. Prozeller, "Management architecture specification", TINA-C, December 1993, www.tinac.com

[11] WBEM Initiative, http://www.dmtf.org/wbem/index.html

[12] Windows Management Instrumentation, Microsoft website, http://msdn.microsoft.com/developer/sdk/wmisdk/default.asp

[13] Sharon Fisher, Policy Management: An introduction, Gartner Group/Datapro Technical Report, February 5, 1999

[14] Nomura, Y, Chugo, A, Adachi, M., Toriumi, M., A Policy Based Networking Architecture for Enterprise Networks, IEEE International Conference on Communications, 1999

[15] Maullo, M.J.; Calo, S. B., Policy management: an architecture and approach, Systems Management, 1993, Proceedings of the IEEE First International Workshop

[16] Vardalachos, N., Galis, A., Serrat, J., Garcia, J., Hoekstra, G., "Policy Based Network Management for IP over WDM Networks", IEEE ICT 2001 Conference proceedings, Bucharest, Romania, 4-7 June 2001; ISBN 973-99995-3-0

[17] WINMAN consortium, "Integrated Network Management requirements and architecture for IP transport services", April 2001

[18] Telemanagement Forum, "Telecommunications Operation Map, v2.1", GB910, November 1999

[19] WINMAN consortium, "WINMAN Solution description R0", August 2001

[20] "Multi Technology Network Management Information Agreement, NML-EML Interface, TMF 608", Member evaluation Version 1.0, May 2001

[21] "Connection and Service Management Information Model (CaSMIM) Information Agreement TMF 605", Public Evaluation, Version 1.5, June 2001

[22] Damianou, N., Dulay, N., Lupu, E., Sloman, M., "Ponder: A Language for specifying Security and Management Policies for Distributed Systems, V 2.3", *Imperial College Research Report DoC 2000/1* Oct. 2000.

[23] Damianou, N., Dulay, N., Lupu, E., Sloman, M., "The Ponder Policy Specification Language", *Proc. Policy 2001:Workshop on Policies for Distributed Systems and Networks,* Bristol, UK, Jan. 2001, Springer-Verlag LNCS 1995

[24] CONTEXT site, http://context.upc.es

[25] IETF FORCES Group, (www.ietf.org/FORCES)

[26] Dey, A.K. and Abowd, G.D. (1999). Toward a better understanding of context and context-awareness. GVU Technical Report GIT-GVU-99-22,

College of Computing, Georgia Institute of Technology. <ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf>

[27] Pascoe, J., Ryan, N.S. and Morse, D.R. (1999). Issues in Developing Context-Aware Computing. Proceedings of the International Symposium on Handheld and Ubiquitous Computing (Karlsruhe, Germany, Sept. 1999), Springer-Verlag, pp. 208-221.

[28] Schilit, B.N., Adams, N.I. and Want, R. (1994). Context-Aware Computing Applications. Proceedings of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, pp. 85-90.

[29] Schilit, B.N., Theimer, M.M., Disseminating active map information to mobile hosts, IEEE Network, Volume: 8, Issue: 5, pp. 22 – 32, September 1994

[30] Brown, P.J.; Bovey, J.D.; Chen, X., Context-aware applications: from the laboratory to the marketplace, IEEE Personal Communications, Volume: 4, Issue: 5, pp. 58 – 64, October 1997

[31] Schmidt, A., Beigl, M., Gellersen, H.W., There is more to context than location, Computers & Graphics Journal, Elsevier, Volume 23, No. 6, December 1999

[32] Schmidt, A., K. van Laerhoven, K., 'How to Build Smart Appliances?', IEEE Personal Communications 8(4), August 2001

[33] Tuulari, Esa. Context aware hand-held devices. Espoo, VTT Electronics, 2000. VTT Publications

[34] Chen, G., Kotz, D., A survey of context-aware mobile computing research, Technical Report, TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000

[35] RFC 1902 (MIB structure) , http://www.ietf.org/rfc/rfc1902.txt

[36] RFC 1903 (Textual Conventions) , http://www.ietf.org/rfc/rfc1903.txt

[37] RFC 1904 (Conformance Statements), http://www.ietf.org/rfc/rfc1904.txt

[38] RFC 1905 (Protocol Operations), http://www.ietf.org/rfc/rfc1905.txt

[39] RFC 1906 (Transport Mappings) , http://www.ietf.org/rfc/rfc1906.txt

[40] RFC 1907 (MIB) , http://www.ietf.org/rfc/rfc1907.txt

[41] ASN1 notation, ITU ASN.1, http://www.itu.int/ITU-T/asn1/

[42] IETF Group, http://www.ietf.org

[43] DMTF Group, http://www.dmtf.org

[44] Distributed Management Task Force, Inc., "DMTF LDAP Schema for the CIM v2.5 Core Information Model", April, 2002. Available at the following DMTF webpage: http://www.dmtf.org/standards/documents/DEN/DSP0123.pdf

[45] PCIM extension, http://www.ietf.org/internet-drafts/draft-ietf-policy-pcim-ext-06.txt

[46] COPS Protocol, http://www.ietf.org/rfc/rfc2748.txt

[47] J. Strassner, and S. Schleimer. "Policy Framework Definition Language". IETF Internet-Draft, Nov., 1998. Available at: http://www.ietf.org/proceedings/99mar/I-D/draft-ietf-policy-framework-pfdl-00.txt

[48] J. Nicklisch. "A rule language for network policies". *Proceedings of Policy Workshop 1999.* Bristol, UK, 1999. Available at: http://www-dse.doc.ic.ac.uk/events/policy-99/pdf/26-Nicklisch.pdf

[49] Policy Research Group, Department of Computing, Imperial College London, http://www-dse.doc.ic.ac.uk/Research/policies/index.shtml

[50] N. Damianou, N. Dulay, E. Lupu, M Sloman. "The Ponder Specification Language". *Proceedings of Workshop on Policies for Distributed Systems and Networks (Policy2001)*, HP Labs Bristol, UK, Jan. 2001.

[51] B. Meyer, and C. Popien. "Defining Policies for Performance Management in Open Distributed Systems". In *Proceedings of the IEEE/IFIP Workshop on Distributed Systems Operations and Management (DSOM94)*, Toulouse, France, October 1994.

[52] R. Weis. "Policies in Network and Systems Management - Formal Definition and Architecture". *Journal of Network and Systems Management*, 2(1): 63-83, Plenum Publishing Corp., March 1994.

[53] R. Weis. "Using a Classification of Management Policies for Policy Specification and Policy Transformation". In *Proceedings of the IFIP/IEEE*

International Symposium on Integrated Network Management, California, USA, May 1995.

[54] T. Koch, C. Krell, et al. "Policy Definition Language for Automated Management of Distributed Systems". In *Proceedings of the IEEE Second International Workshop on Systems Management*, Toronto, Canada, June, 1996.

[55] Imperial College Policy Research Group, http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml

[56] Galis, A., Denazis, S., Brou, C., Klein, C. (ed), " Programmable Networks for IP Service Deployment" ISBN 1-58053-745-6; June 2004; Artech House Books

[57] FAIN project website, http://www.ist-**fain**.org

[58] Yang, K., Galis, A., Serrat, J., Jean, K., Vardalachos, N., Guo, X.,"Network-Centric Context-aware Service over WLAN and GPRS Networks", proceedings of the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Beijing, China,7-10 September 2003

[59] Vardalachos, N., Rubio, J., Galis, A., Serrat J., "A Policy Management System for Hybrid Networks", proceedings of the London Communications Symposium, University College London, London, United Kingdom, 8-9 September 2003

[60] Vardalachos, N., Grampin, E., Galis, A., Serrat J., "A Policy Manager for IP over WDM Networks", proceedings of Eurescom 2002; Powerful Networks for Profitable Services, Heidelberg, Germany, 21-24 October 2002

[61] Vardalachos, N., Grampin, E., Galis, A., Serrat J., "Using Policies on Management of Hybrid Networks", proceedings of the London Communications Symposium, University College London, London, United Kingdom, 9-10 September 2002

[62] Grampín, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, A., "Implementation Issues in PBNM Systems", paper published in the magazine Hiradastechnika (in Hungarian).

[63] Vardalachos, N., Grampin, E., Galis, A., Serrat J., "Policy Management Approach for IP over WDM Networks: A Synthesis Study", proceedings of the 6th London Communications Symposium, University College London, London, United Kingdom, 10-11 September 2001

[64] Grampin, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, J., "Implementation issues of policy based network management systems", proceedings of the 3rd International Workshop on Design of Reliable Communication Networks (DCRN2001), Budapest, Hungary, 7-10 October 2001

[65] Garcia, R., Vardalachos, N. Grampin, E., Raptis, L., Karayannis, F., Vivero, J., Serrat, J., "An approach on policy-based management for IP connectivity over WDM networks", 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Orlando, USA, 22-25 July 2001

[66] Raptis, L., Karayannis, F., Serrat, J., Vaxevanakis, K., Galis, A., Arozarena, P., Vardalachos, N., Chatzilias, G., Chronis, D., Garcia, R., Romijn, W., Philipopoulos, P., Patikis, Y., Josef, D., Schwartz, A., Zahariadis, T., "Integrated Management of IP over Optical Transport Networks", proceedings of IEEE International Conference on Telecommunications, Bucharest, Romania, 4-7 June 2001

[67] Vardalachos, N., Galis, A., Serrat, J., Garcia, R., Hoekstra, G., "Policy Based Management for IP over WDM Networks", proceedings of IEEE International Conference on Telecommunications, Bucharest, Romania, 4-7 June 2001

[68] Serrat, J., Galis A., "Deploying and Managing IP over WDM Networks", June 2003; ISBN 1-58053-501-1, Artech House Books

[69] XMLSpy, http://www.xmlspy.com/

[70] XML Software, http://www.xmlsoftware.com/

[71] XML Parsers, http://www.xmlsoftware.com/parsers/

[72] Java XML, http://java.sun.com/xml/download.html

[73] Crimson, http://xml.apache.org/crimson/

[74] XALAN, http://xml.apache.org/xalan/index.html

[75] SNMP Version 1, RFC 1157, http://www.ietf.org/rfc/rfc1157.txt

[76] Larmouth, J., "Understanding OSI", Chapter 8, International Thomson Computer Press 1996, ISBN 1-85032-176-0

[77] Dubuisson, O., "ASN.1 - *Communication between heterogeneous systems*", Morgan Kauffmann Publishers 2000, ISBN: 0-12-6333361-0

[78] DMTF DEN Initiative, http://www.dmtf.org/standards/den

[79] LDAP v3 (RFC 2251), http://www.ietf.org/rfc/rfc2251.txt

[80] Dev, A., "Database-SQL-Rdbms Howto: Postgresql Object Relational Database System", Iuniverse Inc 2000, ISBN 0-59-513675-3

[81] Berndt, H., de la Fuente Millán, L. A., Graubmann, P., "Service and Management Architecture in TINA-C", Communicaciones de Telefonica I+D, Numero 13, junio 1996, Especial CPSA

[82] Hellemans, P., Mampaey, M., Vanderstraeten, H., Zandbelt, H., Han Zuidweg, H., De Ceuleners, P., Mota, T., "Development and Deployment of a Heterogeneous Set of Services Challenging a TINA-Based Telecommunication Architecture", Lecture Notes in Computer Science, Volume 1430 / 1998, Springer-Verlag Heidelberg, ISSN: 0302-9743

[83] The Dolmen Project, http://www.fub.it/dolmen/

[84] Becker, G.E., Rudrapatna, R., Sowlay, S., Wong, K.N., Wu, J.R, "Integrated Network and Element Management System for the 3rd generation CDMA2000 wireless network", IEEE/IFIP Network Operations and Management Symposium, NOMS 2000

[85] Hong-Taek Ju, Mi-Jung Choi, Hyun-Jun Cha, Sook-Hyang Kim, Hong, J.W.-K., "Effective management application interface and integration mechanisms for Web-based network element management", IEEE/IFIP Network Operations and Management Symposium, NOMS 2000

[86] Mi-Joung Choi, Hong-Taek Ju, Hyun-Jun Cha, Sook-Hyang Kim, Hong, J.W.-K, "An efficient embedded Web server for Web-based network element management", IEEE/IFIP Network Operations and Management Symposium, NOMS 2000

[87] Hung, J.L.S., Gutierrez, J.A., "The application of Web-based technologies on integrated network management", Global Telecommunications Conference, 1998. GLOBECOM 1998

[88] Tennenhouse, D., Smith, J., Sincoskie, D., Wetherall, D. and Minden, G., "A Survey of Active Network Research", IEEE Communications Magazine, Vol 35, No1, p80-86, 1997

[89] Tennenhouse, D. and Wetherall,. D.,"Towards an Active Network Architecture",Computer Communication Review, Vol 26, No 2, 1996

[90] Galis, A., Plattner, B., Smith, J., Denazis, S., Moeller, E., Guo, H., Klein, C., Serrat, J., Laarhuis, J., Karetsos, G., Todd C., "A Flexible IP Active Networks Architecture, Active Networks, Second International Working Conference, IWAN 2000.

[91] Yemini, Y. and daSilva, S.,"Towards programmable networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, October 1996.

[92] Thompson, C., "Agent Technology White Paper and RFP Roadmap", OMG Agent Working Group, Draft .04, March 14, 2000

[93] Zhang D., Zorn W., "Developing Network Management Applications in an Application-Oriented Way Using Mobile Agent", Computer Networks And ISDN Systems, 30(16-18), pp. 1551-1557, Sep. 1998

[94] Brachman, R.J., "What's in a concept: Structural foundations for semantic networks," International Journal of Man-Machine Studies 9, pp. 127-152 (1977)

[95] W.A. Woods. "What's in a link: foundations for semantic networks". Reprinted in "Readings in Knowledge Representation," ed. R.J. Brachman, H.J. Levesque.

[96] Sygkouna, I., Vrontis, S., Chantzara, M., Anagnostou, M., Sykas, E., "Context-Aware Services Provisioning on Top of Active Technologies" IFIP 5th International Conference on Mobile Agents for Telecommunication Applications (MATA 2003), 8-10.10, 2003, Marrakech, Morocco.

[97] CONTEXT project, "Design and implementation of components for the proof of concept of provisioning and management of context aware services", Public Deliverable D3.2, http://context.upc.es/deliverables.htm

[98] W3C website, XML schema http://www.w3.org/XML/Schema

[99] PostgreSQL, http://www.postgresql.org/

[100] The DINA platform, v2.0, http://www.cs.technion.ac.il/Labs/Lccn/DINA/

[101] The IETF Mobile Ad-hoc Networks (manet), http://www.ietf.org/html.charters/manet-charter.html

[102] Baker, F.,"An outsider's view of MANET," Internet Engineering Task Force document, 17 March 2002., http://quimby.gnus.org/internet-drafts/draft-baker-manet-review-00.txt

[103] Eurescom Project P918, "Integration of IP over Optical Networks: Networking and Management", http://www.eurescom.de/public/projects/P900-series/P918/default.asp

[104] Blight, D., Czezowski, P., "Management issues for IP over DWDM networks", Optical Networks Magazine, Vol. 2, Issue 1, Jan/Feb 2001

[105] IPRoute2 Utility Suite How-to, http://www.linuxgrill.com/iproute2-toc.html

[106] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel VIllela: "A Survey of Programmable Networks", ACM SIGCOMM Computer Communication Review, April 1999

[107] Open Signalling Working Group, http://www.comet.columbia.edu/opensig/

[108] DARPA Active Network Program, 1996, http://www.darpa.mil/ato/programs/activenetworks/actnet.htm.

[109] Kalaiarul Dharmalingam and Martin Collier: "Netlets: A New Active Network Architecture" First Joint IEI/IEE Symposium on Telecommunications Systems Research, Dublin, Ireland, 27th November 2001

[110] Tennenhouse David, Smith Jonathan: "A survey of Active Network Research", IEEE Communications Magazine, January 1997

[111] Netfilter: http://www.netfilter.org [Referred 13th January 2003]

[112] GSM World: http://www.gsmworld.com/technology/gprs [Referred 19th January 2004]

[113] Cisco – GPRS White Paper: http://www.cisco.com/warp/public/cc/so/neso/gprs/gprs_wp.htm [Referred 19th January 2004]

[114] Galis, A., Denazis, S., Brou, C., Klein, C. (ed) – " Programmable Networks for IP Service Deployment" ISBN 1-58053-745-6; pp450, April 2004 by Artech House Books; www.artechhouse.com

[115] Siptrex, http://www.siptrex.net

[116] Strassner, J., Moore, B., Moats, R., Ellesson, E., "Policy Core Lightweight Directory Access Protocol (LDAP) Schema", RFC 3703, February 2004 (http://www.rfc-archive.org/getrfc.php?rfc=3703)

[117] Cisco DEN Internetworking Technology Handbook documentation, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/diren.htm

[118] Strassner, J., Judd, S.,"Directory-Enabled Networks", version 3.0c5 (August 1998). (http://www.murchiso.com/den/#denspec).

[119] Distributed Management Task Force, Inc., "DMTF LDAP Schema for the CIM v2.5 Core Information Model", April, 2002. Available at the following DMTF web page: http://www.dmtf.org/standards/documents/DEN/DSP0123.pdf

[120] X.500 specification, http://www.itu.int/rec/dologin.asp?lang=e&id=T-REC-X.500-199311-S!!PDF-E&type=items

[121] Mockapetris, P., "Domain Names – Concepts and Facilities", RFC 1034, November 1987, http://www.ietf.org//rfc/rfc1034.txt

[122] RFC 2251, LDAPv3: The specification of the LDAP on-the-wire protocol, http://www.ietf.org/rfc/rfc2251.txt

[123] RFC 2252, LDAPv3: Attribute Syntax Definitions, http://www.ietf.org/rfc/rfc2252.txt

[124] RFC 2253, LDAPv3: UTF-8 String Representation of Distinguished Names, http://www.ietf.org/rfc/rfc2253.txt

[125] RFC 2254, LDAPv3: The String Representation of LDAP Search Filters, http://www.ietf.org/rfc/rfc2254.txt

[126] RFC 2255, LDAPv3: The LDAP URL Format, http://www.ietf.org/rfc/rfc2255.txt

[127] RFC 2256, LDAPv3: A Summary of the X.500(96) User Schema for use with LDAPv3, http://www.ietf.org/rfc/rfc2256.txt

[128] RFC 2829, LDAPv3: Authentication Methods for LDAP, http://www.ietf.org/rfc/rfc2829.txt

[129] RFC 2830, LDAPv3: Extension for Transport Layer Security, http://www.ietf.org/rfc/rfc2830.txt

[130] RFC 3377, LDAPv3: Technical Specification, http://www.ietf.org/rfc/rfc3377.txt

[131] RFC 2307, Using LDAP as a Network Information Service, http://www.ietf.org/rfc/rfc2306.txt

[132] RFC 1779, A String Representation of Distinguished Names, http://www.faqs.org/rfcs/rfc1779.html

[133] RFC 2294, Representing the O/R Address hierarchy in the X.500 Directory Information Tree, http://www.ietf.org/rfc/rfc2294.txt

[134] Razz, D., Shavitt, Y., "An Active Network Approach for Efficient Network Management", IWAN '99, July 1999, Berlin, Germany

[135] Razz, D., Shavitt, Y., "Active Networks for Efficient Distributed Network Management", IEEE Communications Magazine, 38(3), pp.138-143, March 2000

[136] RFC 3535 - Overview of the 2002 IAB Network Management Workshop, ftp://ftp.rfc-editor.org/in-notes/rfc3535.txt

[137] Murch, R., "How Policy-Based Computing can automate tasks and reduce costs", Professional Technical Reference, Prentice Hall, 14 Jan 2005, http://www.phptr.com/articles/article.asp?p=361809&rl=1

[138] Marshall, I.W, Roadknight, C.M, "Adaptive management of an active services network" BT Technol J special issue on "Biologically inspired computing", 18, 4, pp78-84 October 2000

A.1 Appendix

A1.1 CAPL v0.1

This section presents version 0.1 of the CAPL Policy language, which was designed and implemented by the author. This work is described in detail in the author's transfer thesis. A portion of this work is presented here as well, for completeness and to help the reader follow the evolution of the CAPL language.

A1.1.1 <!ELEMENT NETWORK (NAME,KEY,PHYSLOC* , SCHEME,OFTYPE*, MODEL*,DOMAIN)>

This tag is used to describe the possible network allowed for a user. This is consisted of the name of the network, the key required to join, the physical location of the network, the scheme to be used, the type of the network, the model of the network and finally the domain used.

A1.1.2 <!ELEMENT NAME (#PCDATA)>

This tag is used to name the network allowed for a user. This can be the domain name of the network or any other identifier for the specific network.

A1.1.3 <!ELEMENT KEY (#PCDATA)>

This tag is used to pass the encryption key to be used for joining the network. This could either be the GPRS password or the encryption key used for a wireless LAN.

A1.1.4 <!ELEMENT PHYSLOC (LATITUDE,LONGITUDE)>

This tag is used to pass the physical location of the network. This includes the latitude and the longitude of the location.

A1.1.5 <!ELEMENT OFTYPE (#PCDATA)>

This tag is used to describe the type of the network to be joined. The probable values for this one could be anything, but in this version of CAPL it can only be GPRS or WiFi.

A1.1.6 <!ELEMENT SCHEME (#PCDATA)>

This tag is used to describe the scheme/profile to be used in order to join the specific network. Several schemes/profiles could be defined for each of the users at service creation, so this tag is general.

A1.1.7 <!ELEMENT MODEL (ADDRSPACE*, ADDRALLOC*, CONNECTSYSTEM*)>

This tag is used to describe the model of usage of the network. This includes the type of address space, the address allocations scheme and how the system is connected. The ADDRSPACE tag can be either private or public, the ADDRALLOC can be static or DHCP and finally the CONNECTSYSTEM can be fully masquareded or not masquareded.

A1.1.8 <!ELEMENT LATITUDE (#PCDATA)>

This tag is used to describe the latitude of the network to be joined. This is expressed in the form of (N or S) (Number) x (Number) depending on the actual latitude of the network.

A1.1.9 <!ELEMENT LONGITUDE (#PCDATA)>

This tag is used to describe the longitude of the network to be joined. This is expressed in the form of (E or W) (Number) x (Number) depending on the actual longitude of the network.

A1.1.10 <!ELEMENT ADDRSPACE (#PCDATA)>

This tag is used to describe the type of address space. This can be either private or public.

A1.1.11 <!ELEMENT ADDRALLOC (#PCDATA)>

This tag is used to describe the type of address allocation. This can be either done staticaly or through the use of the Dynamic Host Configuration Protocol (DHCP).

A1.1.12 <!ELEMENT CONNECTSYSTEM (#PCDATA)>

This tag is used to describe how the system is connected. This can be fully masquareded[53] or not masquareded.

A1.1.13 <!ELEMENT DOMAIN (DNAME* ,ADDRESS*, USERS*, SNMPmib+)>

This tag is used to describe the authoritative domain of the system. This includes information on the domain name of the owner, the address, the allowed users and finally the SNMP MIB used.

A1.1.14 <!ELEMENT DNAME (#PCDATA)>

This tag is used in order to define the domain name of the network to be joined. This is the authoritative domain name of the owner of the network.

A1.1.15 <!ELEMENT ADDRESS (DEPT*, STREET*, CITY*, COUNTRY*)>

This tag is used to describe where the owner of the network (and probably the network itself) is located/register. This includes the department name, the street, the city and finally the country.

A1.1.16 <!ELEMENT USERS (USER+)>

This tag is used to describe the users allowed to join the specific network.

A1.1.17 <!ELEMENT SNMPmib (mibNAME, OID, CMD)>

This tag is used to describe the SNMP MIB used in the network. This includes information such as the MIB name (mibNAME), the object identifier (OID) that will be used for the configuration and finally the command (CMD) required to be invoked.

---

[53] This is is also known as *Network Address Translation* (NAT). NAT describes the process of modifying the network addresses contained with datagram headers while they are in transit. IP masquerade is the name given to one type of network address translation that allows all of the hosts on a private network to use the Internet at the price of a single IP address.

A1.1.18 <!ELEMENT DEPT (#PCDATA)>

This tag is used to describe the department name to which the network belongs to.

A1.1.19 <!ELEMENT STREET (#PCDATA)>

This tag is used to describe the street in which the network owner can be found.

A1.1.20 <!ELEMENT CITY (#PCDATA)>

This tag is used to describe the city in which the network owner can be found.

A1.1.21 <!ELEMENT COUNTRY (#PCDATA)>

This tag is used to describe the country in which the network owner can be found.

A1.1.22 <!ELEMENT USER (#PCDATA)>

This tag is used to describe the users which are register to use the network. This is mainly a list of allowed usernames.

A1.1.23 <!ELEMENT mibNAME (#PCDATA)>

This tag is used to give the SNMP MIB name used in the network.

A1.1.24 <!ELEMENT OID (#PCDATA)>

This tag is used to describe the SNMP object id of the element used to configure the network.

A1.1.25 <!ELEMENT CMD (#PCDATA)>

This tag is used to describe the command used in order to configure the network appropriately.

A1.2 Policy Formulation using CAPL v0.1

A simple policy in CAPL v.01 can be now authored. This one has been written in order to allow a user to continue having a VPN connection to the company's server when switching from the GPRS connection to the wavelan connection. Assuming that:

- the network name is 'context'
- the key to join it is '00-9D-6B-AF-45-0C-F0-34-7F-97-98-75-31'
- the physical location is N50x52, E4x22
- it is a wireless network
- the user scheme is 'super1'
- the addressing used is private and fully masqueraded
- DHCP is used
- The network belongs to the Electronic Engineering Department of UCL and is located in 66-72 Gower str, London, UK
- The only registered user is nvardala
- The SNMP MIB used is ucl-mib, the object used is .1.3.6.1.4.1.4680.13.253.4.4 and the command required is /sbin/cardctl.

By using the schema described in the previous section, a policy can be written as follows:

```
<?xml version="1.0" ?>
<!DOCTYPE NETWORK SYSTEM "network.dtd" >
<NETWORK>
  <NAME>context</NAME>
  <KEY>00-9D-6B-AF-45-0C-F0-34-7F-97-98-75-31</KEY>
  <PHYSLOC>
    <LATITUDE>N50x52</LATITUDE>
    <LONGITUDE>E4x22</LONGITUDE>
  </PHYSLOC>
  <OFTYPE>WiFi</OFTYPE>
  <SCHEME>super1</SCHEME>
  <MODEL>
    <ADDRSPACE>Private</ADDRSPACE>
    <ADDRALLOC>DHCP</ADDRALLOC>
    <CONNECTSYSTEM>Fully Masqueraded</CONNECTSYSTEM>
  </MODEL>
    <DOMAIN>
```

```
        <DNAME>UCL</DNAME>
        <ADDRESS>
                <DEPT>Electronic Engineering</DEPT>
                <STREET>66-72 Gower str</STREET>
                <CITY>London</CITY>
                <COUNTRY>UK</COUNTRY>
        </ADDRESS>
        <USERS>
                <USER>nvardala</USER>
        </USERS>
        <SNMPmib>
                <mibNAME>ucl-mib</mibNAME>
                <OID>.1.3.6.1.4.1.4680.13.253.4.4 </OID>
                <CMD>/sbin/cardctl </CMD>
        </SNMPmib>
  </DOMAIN>
</NETWORK>
```

A1.3 Testbed Setup and System in Action (CAPL0.1)

The aforementioned policy was tested using the testbed described in Figure 39. It comprises of one laptop and two active routers.

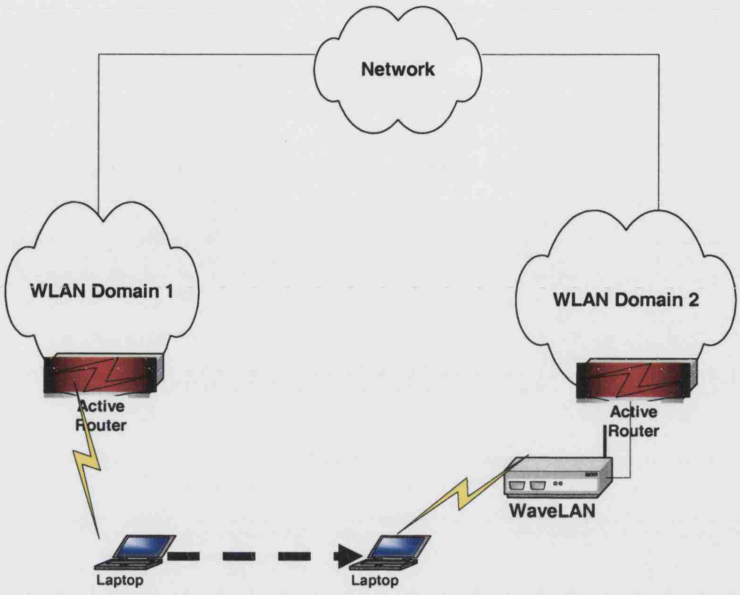


Figure 39: Testbed setup for CAPL scenario

There are two different WLAN domains setup, one simulating the GPRS domain (WLAN Domain 1) and another one (WLAN Domain 2) which is a WLAN hotspot the user will try to join. These two domains are interconnected through a third party network and are routed using active routers. As described in the scenario section, Katherine migrates from her GRPS connection to a WLAN one.

Initially, Katherine is only registered to a GPRS Service. This allows her to connect to WLAN 1. Her service also allows her to connect to her company's server and have access to her files at work by using a VPN. This is done by using an IP GRE tunnel from the GPRS edge router to her company's edge router. By this way she can transfer files to her laptop en route to the hospital (WLAN 2) to see her son. As soon as she arrives at the hospital, her laptop picks up the wireless network offered by the hospital but she cannot connect to that network since she has not registered to the WLAN service as well. So, as she moves within range of the WLAN 2 nothing happens. The WLAN's Service Set Identifier[54] (SSID) can be picked up by a WLAN monitoring utility, but she cannot join that network.

[54] An SSID is the name of a wireless local area network (WLAN). All wireless devices on a WLAN must employ the same SSID in order to communicate with each other.

Eventually, she calls her Service Provider and adds the WLAN service to her existing GPRS account. This service allows her to use wireless networks registered with her provider as soon as she is detected by them. The Service Provider adds a policy[55] allowing her to be able to join WLANs. As soon as the policy is compiled and stored in the policy repository, it is published on the Service Provider's Secure Web Server. She is therefore now allowed to use WLAN 2.

Now, as she moves within range of the WLAN 2 and picks up the SSID of the WLAN, she can use her GPRS connection to connect to her Service Providers Web Server and check if she is allowed to connect to it. If she is, she can download the special file created by the policy. This is an active packet which she can fire into the network. This active packet creates a new VPN for Katherine from the hospital WLAN edge router to her company's server and at the same time tears down the old VPN.

A1.4 Policy Core Information Model Details

The following sections present the different objects included in the Policy Model and a description of the attributes contained by each one of them.

A1.4.1 PolicySet Object

PolicySet		
Attributes	Syntax	Description
policySetId	String	Identification of the Policy Set. This id is used to unambiguously address or refer to this policy set.
policySetAim	String	Aim of the Policy Set. The parameter specifies if this

[55] The policy used in this scenario is the one described in section A1.2.

		policy set is a new one or a modification of a previous one.

A1.4.2 PolicyGroup Object

PolicyGroup		
Attributes	Syntax	Description
policySetId	String	Identification of the Policy Set to which the policy group belongs.
policyGroupId	String	Identification of the Policy Group. This id is used to unambiguously address or refer to this policy group.
policyGroupAim	String	Aim of the Policy Group. The parameter specifies if this policy group is a new one or a modification of a previous one.
policyGroupisAtomic	boolean	This field, dedicated to the management of the policy gorup, defines the way to enforce the policies associated to this policy gorup. It is formed by a boolean value that establishes whether

		concurrent or atomic execution is followed. If the execution is specified as atomic, the policies of the group must to be enforced before starting the evaluation of the next policy group in the sequence.
policyGroupSequencePosition	Int	Contains the position of the policy group within the rest of groups associated to the same Policy Set (first, second, third,...). This position is used when the groups have to be enforced sequentially (in atomic way).
numberofPolicies	Int	Contains the current number of policies associated to this Policy Group.
status	Int	Specifies the current status of the process of this Policy Group. The status of the group refers if the system is waiting for a new policy of the group after have been enforced all the previous ones in the

		sequence (waitingforpolicy (0)); or if the system is waiting the confirmation of the enforcement of the policy currently under evaluation in order to start the evaluation of the following policy in the sequence (waitingforconfirmation (1)); or if all the policies of the group has been successfully enforced (completed(2)).
enfPs	int[]	List of policies, associated to the group, that have been enforced. They are referred using its sequence number inside the group.
recvPs	int[]	List of policies, associated to the group, received. They are referred using its sequence number inside the group. They are referred using its sequence number inside the group.
remPs	int[]	List of policies, associated to the group, which have been erased or removed. They are referred using its

		sequence number inside the group.
sendPs	int[]	List of policies, associated to the group, forwarded to be processed. They are referred using its sequence number inside the group.

A1.4.3 Policy Object

Policy		
Attributes	Syntax	Description
policySetId	String	Identification of the Policy Set to which the policy belong.
policyGroupId	String	The identification of the Policy Group where this policy is included.
policyId	String	Identification of the policy.
policyAim	String	The parameter specifies if this policy is a new one or a modification of a previous one.
policyIsAtomic	boolean	It tells us whether the policy must be atomic or not. It is formed by a boolean value that establishes whether

		concurrent or atomic execution is followed. If the execution is specified as atomic, the policy must to be enforced before starting the evaluation of the next policy in the sequence inside the policy group.
policySequencePosition	int	The order of sequence where you can find this policy within a policy group. Defines the sequence to process the policy within the policy group to which it belongs.
validityPeriod	ValidityPeriod	Is intended to express the policy expiration date. Usually the expiration date is just given with the day and hour when the policy starts and finishes. This attribute is of type ValidityPeriod.
policyEnforcementSequence	String	Specifies the time scheduling (sequential or concurrent) of the policy actions execution. This element must contain a ordened list of actions, including logical words like

		THEN or AND, specifying if the actions must enforced in a sequential or concurrent manner. For example, "Action_1 THEN Action_2" would imply that Action_1 must be enforced before than Action_2, so when Action_1 was successfully enforced, then Action_2 would be enforced. Other example can be "Action_1 AND Action_2", this example implies that Action_1 and Action_2 can be enforced concurrently at the same time.
status	int	Defines the current status of the policy.

A1.4.4 ValidityPeriod Object

ValidityPeriod		
Attributes	Syntax	Description
timePeriod	String	Specifies the validity time period for a policy. This attribute includes the start and end times. Beyond the end time the policy is no

		valid, and will be removed from the system.
Month	String	This parameter is optional, and is used to define a specific time mask.
Week	String	This parameter is optional, and is used to define a specific time mask.
Day	String	This parameter is optional, and is used to define a specific time mask.
Hour	String	This parameter is optional, and is used to define a specific time mask.

A1.4.5 Condition Object

Condition		
Attributes	Syntax	Description
condObject	Vector	This attribute represents the different ConditionObjects that a policy condition can include. The ConditionObjects represent the different variables that will be monitored and evaluated during the

		process the policy. The elements contained in this vector are of type ConditionObject.
condRequirement	Vector	This attribute represents the different ConditionRequirements that a policy condition can include. The ConditionRequirements express the different requirements (e.g. thresholds) to be applied over the values of ConditionObjects in order to decide if the policy condition is fulfilled. The elements contained in this vector are of type ConditionRequirement.
conditionEvaluationMethod	String	Describes how the Condition Requirements specified must be satisfied in order to consider the overall condition fulfilled. This attribute must contain the Condition Requirements identifiers (requirementId) and logical operators (AND, OR,

		NOT). e.g. "(Requirement1 AND Requirement2) OR Requirement3"
conditionEvaluationPeriodicity	String	Express the period of time of the collection and condition evaluation process (for variables requiring time driven measurements).
conditionEvaluationStop	String	Establishes when the evaluation process will be automatically stopped. For instance, it could be specified within this parameter that after having meet the condition one time, or X times the evaluation of the policy will be stopped (that will imply that the monitoration of the variables will be also stopped). If this element is not specified, the policy evaluation will be maintained until the policy will expire, or be removed from the system, or stopped as a result of the enforcement an action.

A1.4.6 ConditionObject Object

ConditionObject		
Attributes	Syntax	Description
Id	String	Specifies the identifier of the ConditionObject.

A1.4.7 MonitoringElement Object

Monitoring element is defined as an abstract class specifying only methods that must implement the child classes that extend this one.

MonitoringElement		
Attributes	Syntax	Description
--	--	--

A1.4.8 Event Object

Event		
Attributes	Syntax	Description
eventType	String	Specifies the type of event to be monitored (e.g. "new_user_in_coverage_area").
evenVar	Vector	Represents an array with the variables of the event. The elements contained in this vector are of type EventVariable.
monCompCompName	String	Represents the name of the

		Monitoring Component entrusted to monitor or listen the event.
monCompCompLoc	Vector	Represents an Array containing locations where the Monitoring component can be found. The elements contained in this vector are of type String.
monPar	Vector	Represents an array with Monitoring Parameters. This attribute is optional and contains any additional information considered useful in reference with the monitoration of the event and that have to be supplied to the monitoring component in order to perform the monitoration activity (e.g. "SSID_name"). The elements contained in this vector are of type Parameter.

A1.4.9 EventVariable Object

EventVariable		
Attributes	Syntax	Description
variableType	String	Represents the type of the event variable. (e.g. "new_user_MAC_address")

syntax	String	Represents the syntax type of this variable.
value	Object	This attribute contains the current value of this Variable.

A1.4.10 SimpleVariable Object

SimpleVariable		
Attributes	Syntax	Description
variableType	String	Represents the type of the simple variable. (e.g. "PacketLoss").
syntax	String	Represents the syntax of this variable.
monCompCompName	String	Represents the name of the monitoring component entrusted to monitor the variable.
monCompCompLoc	Vector	It is an array, which contains Strings indicating the location where the monitoring component can be found. The elements contained in this vector are of String
monPar	Vector	Represents an array with Monitoring Parameters.

		This attribute is optional and contains any additional information considered useful in reference with the monitoration of the variable and that have to be supplied to the monitoring component in order to perform the monitoration activity (e.g. "Router_Interface_Id"). The elements contained in this vector are of type Parameter.
value	Object	This attribute contains the current value of this Variable.

A1.4.11 AggregatedVariable Object

AggregatedVariable		
Attributes	Syntax	Description
Syntax	String	Represents the syntax of this variable.
Operation	String	Represents the operation that will be used to obtain the value of the aggregated variable.
value	Object	This attribute contains the

		current value of this Variable. e.g. "(Variable1) + (Variable2)"

A1.4.12 ConditionRequirement Object

ConditionRequirement		
Attributes	Syntax	Description
requirementId	String	
requirementType	String	It specifies the type of requirement to be applied and evaluated. Possible values are: -Max_Threshold -Min_Threshold -Match_value -In_Margins -Out_Margins -Comparison
reqPar	Vector	This attribute is specific of the type of requirement (for instance, if the requirement consists on apply a minimum threshold; the ReqPar must contain the value of this threshold). The elements contained in this

		vector are of type Parameter
evalComp	EvaluatorComponent	This attribute contains specific information about the component entrusted to evaluate this Condition Requirement. The type of this attribute is EvaluatorComponent.
targetEventId	Vector	If the requirement has to be applied over an Event Variable, this attribute will identify the event to which it belongs. So, this element specifies an event that contains the variable over which the requirement must be applied. The elements contained in this vector are of type String.
targetEventVariableType	Vector	If the requirement has to be applied over an Event Variable, this attribute will identify it. So,this attribute specifies the event variable over whose values the requirement must the applied (This variable belongs to the event identified in targetEventId attribute). The elements

		contained in this vector are of type String.
targetSimpleVariableId	Vector	If the requirement has to be applied over a Simple Variable, this attribute will identify it. The attribute specifies the variable over whose values the requirement must be checked. The elements contained in this vector are of type String.
targetAggrVariableId	Vector	If the requirement has to be applied over an Aggregated Variable, this attribute will identify it. The attribute specifies the variable over whose values the requirement must be checked. The elements contained in this vector are of type String.

A1.4.13 EvaluatorComponent Object

EvaluatorComponent		
Attributes	Syntax	Description
name	String	Represents the name of the evaluator component

		entrusted to evaluate a Condition Requirement.
ocation	Vector	It is an array, which contains Strings indicating the location where the evaluator component can be found. The elements contained in this vector are of type String.

A1.4.14 Action Object

Action		
Attributes	Syntax	Description
actionId	String	Represents the identificator of the Action. e.g. "Action1".
actionType	String	Specifies which specific action has to be enforced (e.g. "create_route").
acParam	Vector	This array contains the different action input parameters to be used when invoking the action. The elements contained in this vector are of type ActionParameter.
enforcerComponentName	String	Represents the Name of the enforcer component entrusted to perform the

		action.
enforcerComponentLocation	Vector	Represents the places where the enforcer components can be located.
enforcementTimeOut	String	Specifies the maximum time available to enforce the action before considering the enforcement as failed.
actionResultValue	Vector	Includes the expected values of the action results in order to consider the action successfully enforced. The elements contained in this vector are of type String.
outputParameter	Vector	Represents the parameter values expected to be returned after the enforcement of the action. The elements contained in this vector are of type Parameter.

A1.4.15 ActionParameter Object

ActionParameter		
Attributes	Syntax	Description
name	String	Specifies the name of the

		parameter.
syntax	String	Specifies the syntax type of the parameter's values.
value	Object	If the value of the action parameter is fixed, it is specified in this attribute.
eventId	String	The current value of a ConditionObject can be used to be assigned to an action parameter. If the current value of an event variable will be used as the value of this action parameter, this attribute will specify the identifier of the event to which the event variable belongs.
eventVariableType	String	The current value of a ConditionObject can be used to be assigned to an action parameter. If the current value of an event variable will be used as the value of this action parameter, this attribute will specify the identifier of this event variable.
simpleVariableId	String	The current value of a ConditionObject can be

		used to be assigned to an action parameter. If the current value of a Simple Variable will be used as the value of this action parameter, this attribute will specify the identifier of this simple variable.
aggrVariableId	String	The current value of a ConditionObject can be used to be assigned to an action parameter. If the current value of an Aggregated Variable will be used as the value of this action parameter, this attribute will specify the identifier of this aggregated variable.

A1.4.16 Parameter Object

Parameter		
Attributes	Syntax	Description
name	String	Represents the name of the parameter.
syntax	String	Represents the syntax type of the parameter.
value	String	Represents the value of the parameter.

A.2 Policy-based Management Paradigm in CONTEXT

A2.1 Rationale – Automating the Process of Service Provisioning

This section describes the Policy-based Management System developed for the CONTEXT project and it's use after the Authoring and Customisation phases. This system executes all the logic operations needed to control the CAS lifecycle. All these operations will be performed using policies, in order to manage the actions to be executed in the Execution Environment Layer to complete all management objectives.

A2.1.1 Why Policies?

A policy is an administrator-specified directive that manages and provides guidelines for how the different network and service elements should behave when some conditions are met. However the answer to the question, "why policies should be used?" is a justification itself and explains why the IST-Context consortium decides to use policies as the main engine to create, deploy and manage context-aware services.

The main benefits from using policies are improved scalability and flexibility for the management system. *Scalability* is improved by uniformly applying the same policy to large sets of devices and objects, while *Flexibility* is achieved by separating the policy from the implementation of the managed system. Policies can be changed dynamically, thus changing the behaviour and strategy of a system, without modifying its implementation or interrupting its operation, Policy-based management is largely supported by standards organizations as IETF and DMTF and most network equipment networks.

Another benefit from using policies when managing is their *Simplicity*. This simplicity is achieved by means of two basic techniques: Centralized Configuration, elements don't have to be configured individually; and Simplified Abstraction, which means that one does not have to configure exactly each device, but only establish the required policy for the overall system to follow and the system will translate this policy and will enforce it in the correct component.

As it will be explained in this later on, the policies are going to be pre-defined in XML; XML was chosen because it is a common language, standardized and versatile. When a user subscribes to an available CAS, then policies are going to be personalised taking these pre-defined policies as a template and taking into account the user's preferences and context.

It is proposed the use of XML as the language to express policies and use the proposed information model and architecture to manage the services based on these policies. The main advantage of using XML as policy language is its flexibility to define and exchange policies wrote in this format.

A2.1.2 Objectives

The main objective for using policies for service management is the same as with managing networks with policies: automated management and perform it at an as high level as possible. The philosophy for managing a resource, a network or a service with a policy-based managed approach is that IF something happens THEN the management system is going to take an action.

Policies can be tailored to different users. The main idea is to use generic policies that can be customized, following user subscription; the parameters of the conditions and actions in the policies are different for each user, reflecting its personal characteristics and its desired context information.

Within CONTEXT system, policies are used to express and subsequently create in a dynamic manner the logic of context-aware services. This is undertaken by the activities of the CAS Creation sub-system.

In addition, policies are used in managing various aspects of the created context-aware services. An important aspect of policy-based service management is the deployment of services throughout the Active Network. For instance, when a context-aware service is going to be deployed over the active nodes in the network, just after user subscription, the decisions that have to be taken in order to determine in which nodes the service is going to be installed is based on the policies customized according to the values introduced by the user and its desired context or environment. Furthermore, context-aware service invocation and execution is also controlled by policies, for determining in a flexible way

when and how customized services will be invoked and executed depending on different aspects. Finally, the maintenance of the code realizing the logic of context-aware services and the assurance of context-aware service operation is also subject to related management policies.

The above concepts have driven our decision for using pre-defined policies expressed in XML, which can be afterwards personalized by user subscriptions to the context-aware services. The solution for the Policy-Based Service Management of the IST-Context Project must rely on a robust and flexible architecture capable of accommodating the service management systems and like correspondence new types of services. The architecture should exhibit the following logical attributes:

- Open:

  It is built on standard interfaces between architectural components.
- Flexible:

  Supports the incorporation of any new context services that complies with the specified interfaces, and allows the modification of the system behaviour by means of user defined policies.
- Modular:

  Its architecture is based on components, standardized and oriented to context services.
- Scalable:

  Achieved with the separation in different building blocks of technology specific and technology independent functionality, and through of the minimization of data redundancies.
- Distributed:

  The selection of a component based architecture defined to run on top of a standards-based object request broker guarantees the transparency of the system to the location of components.

A2.1.3 CONTEXT Policy-Based Service Management System

After introducing the framework that is going to be the basis of the design, the required approach for the Service Management Layer Architecture can be described. Concepts of the generic policy-based management system will be applied in order to solve and implement the functionality described in the IR3.1 for the Service Management Layer.

At the moment of starting the designing the Policy Based Service Management Layer for CONTEXT project, the different functional blocks identified in the document IR3.1 were used as a reference. The result of the Service Creation phase would be the Service Code (Java) and the Management Policies (XML) which are going to manage the provisioning and the maintenance of this service. As the work consists of thinking and making a proposal about the Policy Based Service Management Layer, the Service Creation Phase should remain as it was presented in IR3.1, as the Service Management layer is only interested in its results when talking about the interaction between the two layers or phases.

Regarding Service Management Layer, there are four main functional blocks identified. They are:

- Code Distribution and maintenance
- Code Execution
- Service Invocation
- Service Assurance

They are detailed in the IR3.1. These functional blocks must be policy driven. This means that their functionality must be influenced in a policy ruled way. The next step is to design a policy management system that handles, manages and applies the service management policies that will rule the behaviour of the system, and particularly the efficient delivery of the functionalities identified. This step obviously includes the proposal of the classes or types of policies that will be managed by the system and the component of this system.

The CONTEXT proposal is based on the generic policy-based management system presented. As presented in the previous section, the components that are technology and policy specific are the Action Consumers and the Condition Evaluators. These

components are the ones in charge of understanding the particular policy semantics, monitor its conditions (CE) and apply its actions (AC). For this reason is reasonable that in order to provide the functionalities described in IR3.1, service specific Action Consumers and Condition Evaluators should be used.

The policies proposed will be closely tied with these functionalities. Taking into account these, the first approximation to possible implementation architecture could be the following showed in Figure 40.

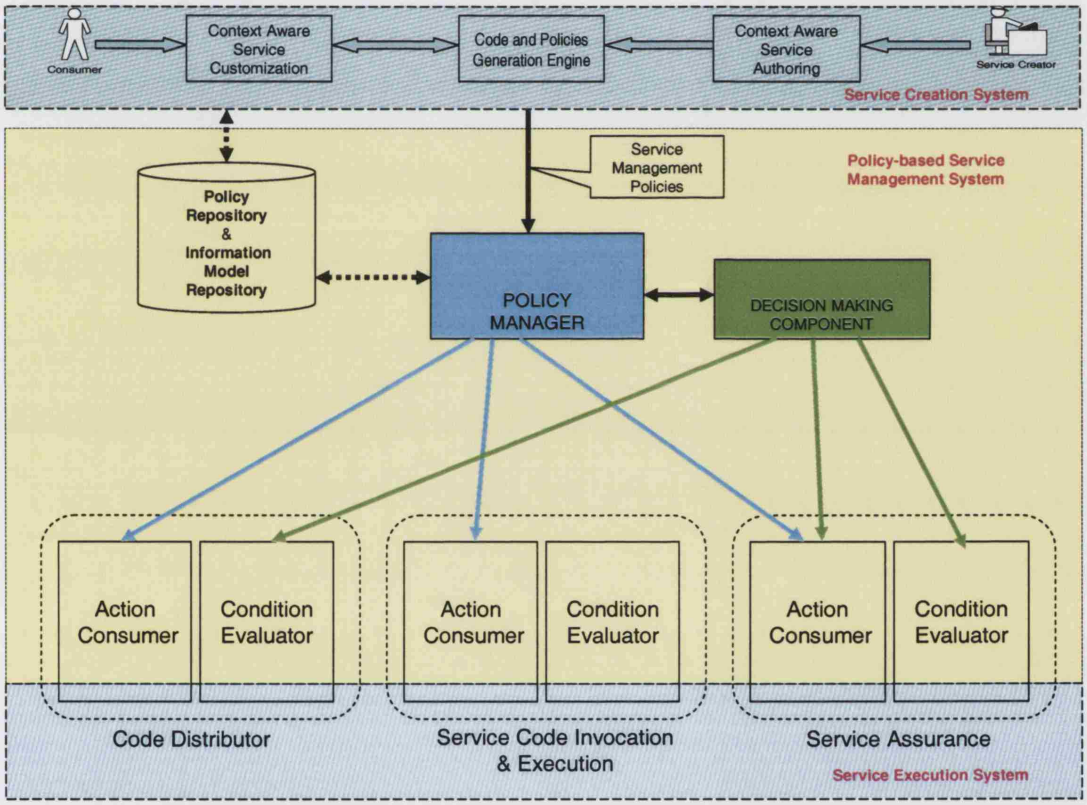


Figure 40: *First approximation of the generic policy-based management system to the CONTEXT service layer functional requirements*

Depending on the functionalities identified to be required the appropriate AC and CE should be designed and implemented. The ACs and CEs are grouped taking into account the functional components they are related to.

The proposed AC and CE functionalities are:

⇨ • Code Distributor *Code Distributor Action Consumer* and *Code Distributor Condition Evaluator.*

⇨ • Code Execution Controller *Code Execution Controller Action Consumer*

⇨ • Invocation Service Listener *Service Invocation Condition Evaluator*

⇨ • Service Assurance *Service Assurance Condition Evaluator* and *Service Assurance Action Consumer.*

It should be taken into account that the Condition Evaluators proposed could not be a unique component but a set of different components specialized in to particular tasks or different instances of these components installed in different points of the network. For example, for service assurance could be different Condition Evaluators instances each one of them specialized in monitoring different performance or quality parameters, or installed in different nodes. Other example could be that for service invocation could be different Condition Evaluators each one of them specialized in monitoring or listening to different kinds of invocation signals or events and installed in different points.

Finally, the proposed architecture for the Service Management layer can be seen in Figure 41.

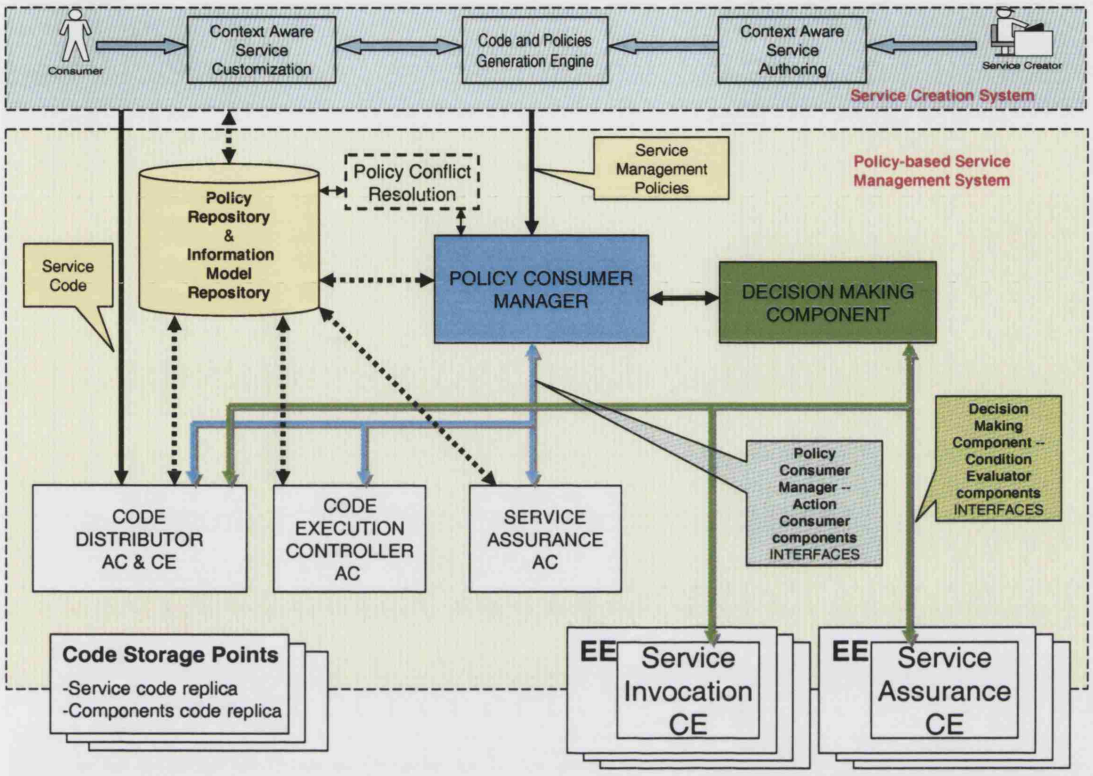


Figure 41: *IST-CONTEXT Policy-Based Service Management Layer Architecture*

This architecture is the result of the adaptation of the Policy-Based Management System described in section 'Policy-Based Management System' to the necessities and functional requirements of the CONTEXT project.

The specific CONTEXT components introduced in the architecture playing the role of Action Consumers (AC) or Condition Evaluators (CE) or giving support to CAS management functional blocks are slightly described in the following sections.

A2.1.3.1 Code Distributor Action Consumer

The Code Distribution Action Consumer (CDAC) is the specific components used for distributing the Service Code. This component is going to enforce the actions, related to the code distribution policies, that have been sent by the PM to be enforced by this AC. The result of the enforcement of these actions will be the distribution of the service code following specific configuration parameters or Code Storage Point selection criteria

included in the policy action. This component must be aware of network capabilities information in order to perform its actions, so it can communicate with the Information Repository to obtain the needed information.

The Code Distributor Action Consumer will run outside the active nodes. This is because the Code Distributor Action Consumer is not to be constrained or to be installed and executed in any specific place in the network. Also, its actions are not reduced to the scope of any particular active node, but these actions can comprise many nodes (Storage Points). The Code Distributor was be implemented as a Java class and was used internally by the Policy Manager when needed. The Code Distributor will implement an API to be used by the Policy Manager to invoke the different code distribution related actions contained in the Code Distribution Policies.

A2.1.3.2 Code Execution Controller Action Consumer

The Code Execution Controller Action Consumer (CECAC) is the specific component thought to control the execution of the Service Code. The actions that are going to be enforced by these consumers are related to when and where to execute the service code. For example, after receiving an invocation signal the actions that are going to start the service execution in the appropriate execution points will be executed by this Action Consumer.

The Code Execution Action Consumer will run outside the active nodes. This is because the Code Execution Controller Action Consumer is not to be constrained or to be installed and executed in any specific place in the network. Also, its actions are not reduced to the scope of any particular active node, but these actions can comprise many nodes (Execution Points). The Code Execution Controller will be implemented as a Java class and will be used internally by the Policy Manager when needed. The Code Execution Controller implemented an API which was used by the Policy Manager to invoke the different code execution related actions contained in the Code Execution Controller Policies.

A2.1.3.3 Service Assurance Action Consumer

Once the service code has started its execution, it is time to proceed to its Assurance. The Service Assurance Action Consumer (SAAC) executes the actions related to the Service Assurance, which will be invoked by reaching certain performance, or assurance conditions that are going to be evaluated by Service Assurance Condition Evaluators.

The Service Assurance Action Consumer will run outside the active nodes. This is because the Service Assurance Action Consumer is not to be constrained or to to be installed and executed in any specific place in the network. Also its actions are not reduced to the scope of any particular active node, but these actions can comprise many nodes.

A2.1.3.4 AC Data API

The different AC must share diverse runtime information, which must be stored somewhere in the system. To provide a common interface as well as information integrity, the AC DATA API is provided.

Its goal is to store, maintain and delete information based on explicit requests of the different ACs. There exist four different data tables, and a consequent handler interface for each of them.

These information tables are the following:

CODE_PROCESS Table:

It registers information of the current execution processes over the DINA network. It contains the following information:

- **CodeID**
- **DINASessionID**
- **DINASeqNumber**
- **DINANodeID** **(multiple)**
- **DINANodeIPAddress** **(multiple)**

This table is managed by the *codeProcessHandler* interface, which will be specified later in the document.

CODE_STORAGE Table:

Its goal is to keep track of the different locations where different copies of code are stored. It contains the following information:

- **CodeID**
- **Copy N°**
- **URLList: list of URLs which contain the code**

This table is managed by the *codeStorageHandler* interface, which will be specified later in the document.

DINA_NODES Table:

It stores information relevant to the DINA nodes in the network. It contains the following information:

- **DinaNodeID**
- **DinaNodeIPAddress**
- **Location: physical location of the DINA node**
- **HasList: contains information of the components/devices the DINA node has (i.e. SIPBroker, WLANBroker, WLANmodel=Cisco74, etc)**

This table is managed by the *dinaNodeHandler* interface, which will be specified later in the document.

STORAGE_POINTS Table:

It stores information relevant to the DINA nodes in the network. It contains the following information:

- **storagePointID**
- **storagePointIPAddress**
- **Protocol**
- **Port**
- **BackupLevel**
- **ListofDinaNodes**

This table is managed by the *storagePointHandler* interface, which will be specified later in the document.

The APIs to create, alter, access and delete information from these tables is specified as follows:

CodeProcessHandler API:

```
public interface codeProcessHandler {

  // New process
     public void addNewProcess(String codeId, int sessId, int
                              seqNumber, InetAddress dinaNodeIp);

  // Recover Information
  public InetAddress [] getDinaNodesIPs (String codeId);
  public int getSessionId(String codeId);
  public int getSequenceN(String codeId);

  // Delete
  public void removeProcess (String codeId);
  public void removeProcessFromNode (String codeId, InetAddress
                                        dinaNodeIP)
}
```

Comments about codeProcessHandler API:

- *removeProcess ()* removes the process specified by the codeID from all DINA nodes
- *removeProcessFromNode ()* removes the process only from the specified node (given by the IP or the dinaNodeID)

CodeStorageHandler API:

```
public interface codeStorageHandler {

        // New code
        public void addCode(String CodeId, int copyNumber,
                              String[] URLList);
```

```
    // Recover info
    public String[] getStorageURLs(String CodeId);
    public int[] getCopyNumbers(String CodeId);

    // Delete
    public void removeCode(String CodeId);
  }
```

Comments about codeStorageHandler API:

- *addCode ()* example usage. If CD AC wants to distribute a certain code of a service named "SRV1" which has two classes "main.class" and "aux.class", CD should use this method in a similar way as below:

```
addCode ("SRV1", 1, {"ftpA://code/srv1/main.class",
                     "ftpA://code/srv1/aux.class"});
addCode ("SRV1", 2, {"ftpB://opt/srv1/main.class",
                     "ftpB://opt/srv1/aux.class"});
   addCode ("SRV1", 3, {"ftpC://context/srv1/main.class",
                        "ftpC://context/srv1/aux.class"});
```

NOTE: copyNumber can be seen as a copy identifier, and it should be different for all copies

dinaNodeHandler API:

```
public interface dinaNodeHandler {

  // New node
  public void addDinaNode(String nodeID, InetAddress nodeIP,
                          String location, String [] HasList);



  // Recover info
  public InetAddress[] getDinaNodes(String Location, String []
```

```
                                    FilteringWildcard);
    public InetAddress getDinaNodeIP(String dinaNodeID);
    public String getDinaNodeID(InetAddress dinaNodeIP);


    // Delete
    public void removeDinaNode(String NodeID);
    public void removeDinaNode(InetAddress NodeIP);
}
```

Comments about dinaNodeHandler API:

- HasList passed as parameter in the *addDinaNode ()* method is similar to the following:

```
HasList1 = {WLANbroker=yes, WLANmodel=Cisco62,
SIPBroker=yes}
```

- The FilteringWildCard parameter in the *getDinaNodes()* method is a string array. Each element of the array has the following format:

```
element=attribute
```

eg:

NodeID1 --> HasList1 = {WLANbroker=yes, WLANstd=802.11b, WLANmodel=Cisco62}
NodeID2 --> HasList2 = {WLANbroker=yes, WLANstd=802.11b, WLANmodel=Cisco71}
NodeID3 --> HasList3 = {WLANbroker=yes, WLANstd=802.11b, WLANmodel=Cisco74}

- A wild card shall be represented for example as:

```
WLANmodel=Cisco7* --> Cisco71, Cisco74
```

And the *getDinaNodes ()* method would return:

```
[Node2_IP Node3_IP]
```

storagePointHandler API:

```
public interface storagePointHandler {

  // New storage point
  public void addStoragePoint(String storagePointID, InetAddress
                        stpIP, String protocol, int port, String backLevel,
                        String[] dinaNodeIdList);

  // Recover info
  public InetAddress getIP (String storagePointID);
  public String getProtocol(String storagePointID);
  public int getPort (String storagePointID);
  public String getBackLevel (String storagePointID);
  public String[] getDinaNodeIdList (String storagePointID);

  // Delete
  public void removeStoragePoint(String storagePointID);
}
```

A2.1.3.5 Code Distributor Condition Evaluator

The Code Distributor Condition Evaluator is intended to monitor and evaluate conditions regarding code distribution and maintenance purposes. The conditions regarding code distribution can refer to monitoring events that must trigger some code distribution actions, such as the reception of new service code, etc.

The code maintenance policies are in charge of assuring the maintenance of the service code that is stored at the Code Storage Points. The conditions regarding maintenance can refer to monitoring code expiration events, reception of new code version, etc. The Code Distributor Condition Evaluator will run outside the active nodes.

A2.1.3.6 Service Invocation Condition Evaluator

This Condition Evaluator is in charge of listening and filter service invocation events or signals (e.g. all messages using a specific protocol) in order to enforce the execution of a specific Context-Aware Service. Every different Service and even every user could have

different requirements to be applied over the events or signals captured, which can be specified during the customization phase. The appropriate SICEs will be downloaded to the correct nodes or places for its execution. Afterwards, as a result of the conditions of the policies that manage the invocation of a service, the SICEs will be configured specifying all the invocation events or variables to attend and the filters to apply. The information derived from the invocation events received and the application of the *Condition Requirements* specified should be enough to decide without ambiguity the specific service code to execute.

The Service Invocation Condition Evaluator will run inside the active nodes (EE). The SICEs should be installed and executed in particular nodes where the invocation events and variables must be monitored. Is more efficient to install these components close to the nodes where likely the invocation signals will be received.

A2.1.3.7 Service Assurance Condition Evaluator

The Service Assurance Condition Evaluator is the generic name for the possible different Condition Evaluators that are intended to evaluate the different variables that have been decided to determine the quality of the service. There could be variables applicable to all services and others specific for each one.

Once the DMC has received the conditions of the policies related to Service Assurance, it is going to configure the different Condition Evaluators in the appropriate Active Nodes, where the different assurance parameters have to be monitored. Afterwards, the SACE will obtain these parameters from the running service and will apply the correspondent filters or thresholds. If any parameter fall outside desirable margins, a notification will be send by the SACE to the DMC in order to communicate the event.

The Service Assurance Condition Evaluator will run inside the active nodes (EE). The SACEs should be installed and executed in particular nodes where these events and variables must be monitored.

A2.1.3.8 Policy & Information Model Repository

The Information Model Repository is the logical storage point for the information about Policies loaded in the system, the network where the CAS is going to be provided

(Network Inventory), the services deployed, the components installed, and other information entities (Management Information). Such information contains the status and functionality of every service layer related entity.

This information is of great importance, for example, when evaluating conditions in order to know whether the appropriate Condition Evaluator is already installed or not. Another important utility of the information contained in this repository could be to decide where a Service Code must be stored depending on the capabilities of the different Storage Points available. In the same manner, could be also important to decide where a Service must be executed depending on the capabilities of the Active Nodes and the conditions introduced by the user at the customization phase.

The Service Creation System will have to be aware of the different Action Consumer components and Condition Evaluator components and mainly of the different actions and Conditions they are able to enforce and evaluate, in order to compose the Management Policies.

A2.1.3.9 Code Storage Points

The Code Storage Points are the physical places where the service code is stored after its distribution and before its execution on the Active Nodes. It means that the requested code will be downloaded from these Storage Points.

This Code Storage Points could be able to store also code of management system components such as Condition Evaluators and Action Consumers.

A2.2 Domain-specific Policies

The following section aims to identify the possible CONTEXT Project domain-specific policies that will be used in the implementation phase of the Policy-Based Service Management.

It has been identified Policies grouped into four functional domains, Service Code Distribution, Service Code Maintenance, Service Code Invocation and Execution, and Service Assurance. These are the main policy domains to be developed although future implementations will show the practicability to add or delete new policy types according to the project needs.

The high level description of the policies exposed in this section follows the next format

IF (*condition 1*) [**AND**|**OR** (*condition n*)]

THEN (*action 1*) [**AND** (*action n*)]

The Service Management Policies control just the service lifecycle, never the logic of the service. In this way, Service Management policies are used by the PBSM components of the system to define the Code Distribution and Code Maintenance of the service as well as the Service Invocation, Service Execution and Service Assurance.

The specification of the different policies implied in the different functional phases of the service management will imply the identification of a set of conditions and actions that these policies will contain. It is very important to agree in the specification of the policies used in the PBSMS because all the actions and conditions contained in policies must be implemented in the different components playing the role of Action Consumers and Condition Evaluators.

A2.2.1 Service Code Distribution Policies

These policies govern the distribution of code on the PBSM System Storage Points components. This code refers to the CAS logic. The main points to notify are:

- Will be processed when new customized service code arrives to distribution component.
- These policies drive the deployment process of needed service.
- They specify the specific configuration, required resources and criteria for optimum Storage Point selection for the service.

A2.2.1.1 Service Deployment Policies Group example

In this section a high level example of this type of policies is presented.

Service Code Distribution Policy

1) "**If** (customized service B code is received) **then** (configure distribution of service B code and optimum Storage Point selection parameters)"

Conditions:

This policy condition should be based on an event and appears when a new service code arrives to the Code Distributor component. This event could contain information needed to identify the code just received. The condition is met for this policy if the code arrived is the one of the Service B. The requirements express this fact applying the requirement type "Match_value" to the event variables in order to check if the code received is Service B code. The component carried out to monitor the event and evaluate the requirements is the Code Distributor (Condition Evaluator).

Actions:

The action implies distributing the service B code using the particular configuration expressed through the action parameters specified in the policy. These input parameters will specify how the code has to be distributed. This action is enforced by the **Code Distributor** (Action Consumer).

A2.2.2 Service Code Maintenance Policies

These policies allow the maintenance of the code installed along the infrastructure to support services. The main points to notify are:

- Will be enforced when maintenance event arise. These events regards to new service version, service expiration, storage points resources under desirable margins, high load of invocation petitions, etc.

- The actions enforced will be service code removal, service code update, service code redistribution (changing number of replicas, Storage Point selection criteria, etc).

A2.2.2.1 Service Maintenance Policy Group Example

In this section a high level example of this type of policies is presented. The high level policies included in this group could be the following ones:

1) "**If** (new version of customized service B code)
then (remove old code version of service B from Storage Points)
& (distribute new service B code)"

2) "**If** (customized service B code expiration date has been reached)
then (deactivate execution polices for service B)
& (remove code of service B from Storage Points)"

3) "**If** (The number of invocations for service B is very high)
then (distribute more service B code replicas to new Storage Points)"

4) "**If** (Resources in Storage Point X are under desirable margins for service B)
then (remove service B code from Storage Point X)
& (distribute one replica of service B code to a new Storage Point)"

A2.2.3 Service Code Invocation and Execution Policies

These policies control the monitoring of variables or events that start the execution of a context-aware service that have been subscribed and customized. They will be enforced when a service is invoked. The invocation signal will be used to deduce the service to execute and the execution parameters associated.

Related to the Invocation & Execution policies and form an informational viewpoint, there are three abstraction levels, related to the invocation of a service and the execution of the respective customised code:

a. I1: The 'raw' information coming from the system devices (e.g. servers, routers etc.). This information is input to the SICEs. Usually, this

information is at the lowest abstraction level (e.g. at the level of userId, password, raw data measurement).

b. I2: The information coming from the SICEs to the DM i.e. the variables in the notifications that the SICEs send to the DM. This SICE output information is basically the filtered outcome of the information input to the SICEs. It could be at the abstraction level of the input information (i.e. at the level of I1). However, it could be at a higher abstraction level, if in addition to filters, SICEs would be configured with appropriate *mappers*.

c. I3: The information that identifies (a) which customised service(s) to execute, (b) the initial conditions (run-time arguments) and (c) where to execute the customised service code determined for execution. This information corresponds to the input parameters required by the action undertakers (ACs), which will actually download the code for execution. This information is specific to the CAS creation system, depending on the particular scheme/convention adopted to name the instances of customised code (it should be stressed that this naming is specific to the CAS creation system and not to the PBMS; ideally PBMS should not restrict the naming of code instances). For instance, customised code could be named after the concatenation of subscriptionId and serviceId (or just subscriptionId if we assume that a subscription contains only one services). But, this naming may not be efficient or it may not be suitable at all for a particular CAS, since a subscriber may change its customisations in the context of a subscription or may have a number of customisation options. Therefore, a more efficient or suitable (for some CASs) naming would be to name customised code after the concatenation of subscriptionId, serviceId and customisationId, where the latter indicates the appropriate customisation to apply at a given time.

The point is that the abstractions in levels I1 and I3 are different. The purpose of the service invocation and code execution control process in our CONTEXT system is to go

from abstraction level I1 to I3. That is, based on the 'raw' information (at I1 abstraction level), to determine the triple <subsriptionId, serviceId, customisationId> (or any other tuple that the CAS creation system would adopt for naming CAS code instances), as well as to determine the run-time arguments and the place of execution of the code to be executed.

Broadly speaking, the policy-based operation relying on dynamically defined policies, provides flexibility in going from I1 to I3, thus facilitating service introduction (new CASs) and automated service provisioning (of new customisations). This is the beauty and the strength of the CONTEXT system.

The code invocation and execution domains are not only for deducing the conditions necessitating the execution of a CAS code but also for identifying the code instance to be sent for execution and the associated execution parameters.

Regarding the abstraction levels exposed above, moving from I1 to I3 is conveyed by the nature of the policy: the association of a specific condition to a specific action.

A condition specifies that if some variables (Condition Objects) adopt some specific values (Condition Requirements) then we assume that the condition is fulfilled. We think that the Condition Objects are expressed at the level I1.

This condition will be associated to an action that will execute some specific piece of code with specific initial conditions. The input parameters of the action regards to the I3 level. Keep in mind that the input parameters values of the action can be 'hard-written' since policy edition/creation and other ones can adopt its values from some monitored variable defined at policy condition. So the association between the I1 level and the I3 level is directly derived from the association between the condition (I1) and the action of the policy (I3).

A2.2.3.1 Service Execution Policy Group Example

In this section, a high level example of this type of policies is presented.

Service Code Invocation and Execution Policy

1) **"If** (invocation event X is received) **then** (customized service B must be executed)"

Conditions:

In this example, the policy condition is based on an event (simple or aggregated variables can be also used). This event is called *Invocation_Event_X* and appears when a particular invocation signal X is received. This event has associated different variables that can represent different kinds of information that the invocation signal can contain (for instance, a used_id and a password included in some invocation signal). The condition is met for this policy if the event appears and the event variables associated accomplish the requirements expressed in the condition. The component carried out in the policy to listen to the event and evaluate the requirements is a suitable type of **Service Invocation Condition Evaluator** (capable of monitor the event type X).

Actions:

The action implies executing the Service B using the particularized execution parameters included in the specification of the action parameters. This action is enforced by the **Code Execution Controller Action Consumer.**

A2.2.4 Service Assurance Policies

This type of policies is intended to support the fulfilment of QoS levels established for the different services. Essentially, they are devoted to collect variables and aggregates to compute service quality indexes and inform the system whether these indexes are not kept into the appropriate intervals, and to perform the corrective actions considered. It has been envisaged three different types of policies inside the Assurance Policy Group.

- Service Assurance Initialization Policies

- Processed when a service start its execution. These policies provide the any action needed to configure the running service (SLO) to export assurance parameters.

- Service Assurance Execution Policies.
  - Processed after assurance initialization and during a service execution. Provide the actions to be applied when assurance parameters are within unacceptable margins. These policies can be based on different severity levels, applying different corrective actions depending on them.
- Service Assurance Finalization Policies
  - Processed when a service stops its execution. Provide the any action to be applied in order to stop the assurance activity associated with the stopped service.

It's supposed that the Service Assurance Condition Evaluator and the Service Assurance Action Consumer will be the main components responsible for monitor the conditions and enforce the related actions.

A2.2.4.1 Service Assurance Policy Group Example

In this section a high level example of this type of policies is presented.

Service Assurance Initialization Policies

1) "**If** (customized service B is running)
then (configure assurance parameters for service B) & (configure local assurance variables)"

Service Assurance Execution Policies

2) "**If** (level=2) & (parameterA>X) **then** (Action M)"

3) "**If** (level=2) & (parameterB>Y) **then** (Action N)"

4) "**If** (level=2) & (parameterC<Z) **then** (level=1) & (Action K)"

5) "**If** (level=1) & (parameterA>X) **then** (Action P)"

6) "**If** (level=1) & (parameterD>J) **then** (Action O)"

Service Assurance Finalization Policies

7) "**If** (customized service B is stopped)
then (stop assurance parameters evaluation) & (remove local assurance parameters)"

A2.3 Code Distributor

A2.3.1 Code Distributor Action Consumer API

The CodeDistributor Action Consumer controls the distribution of the service code (SLO/SICE) to the code repositories or directly to the execution points.

The main methods are the following ones:

'DistributeCode' method provides the mechanism to distribute a new SLO/SICE to a set of the code repositories or code execution points (DINA node) depending on the level argument. The level refers to administrative domains eg level 1 as close to the DINA

nodes as possible and level 3 not very close at all. This method also adds the entry for the new code into the code_storage table.

'GetOptimalURLOfCode' method provides the mechanism to retrieve a specific URL of where the SLO/SICE code resides by looking it up in the code_storage table.

'RemoveCode' method provides the mechanism to remove the SLO/SICE from a specific code repository or DINA node along with its corresponding entry in the code_storage table.

```
public class CodeDistributorInterface {

    public CodeDistributorInterface();

    public boolean DistributeCode(String CodeId, URL[] URLList,
int NoCopies, int Level, String[] PotentialExecPoints);

    public URL[] GetOptimalURLOfCode(String CodeId, InetAddress
DINANodeIPAddr);

    public boolean RemoveCode(String CodeId);
```

A2.4 Code Execution Controller AND Service Invocation Listeners

A2.4.1 Code Execution Controller API

The Code Execution Controller allows to execute active services and to send arguments to existing active services.

'executeCode' trigger an execution of a service that is identified by the *'codeId'* and pass the arguments *'arg'* to the service. The code can be executed on a specific node (*'node'*)

or on a set of nodes that are derived from a wildcard (*'nodeWildcard'*). The methods return '0' on success or '-1' in case of a failure.

'sendMessageToService' sends the message *'msg'* to a running service that is identified by *'codeId'*. The message can be sent to a service that runs on a specific node (*'node'*) or on a set of nodes that are derived from a wildcard (*'nodeWildcard'*). The methods return '0' on success or '-1' in case of a failure.

```
public class codeExecutionController
{
   public codeExecutionController();

   public int executeCode(String codeId, InetAddress node, String [] arg);
   public int executeCode(String codeId, String nodeWildcard, String [] arg);

   public int sendMessageToService(String codeId, InetAddress node, String msg)
   public int sendMessageToService(String codeId, String nodeWildcard, String msg)
}
```

A2.4.2 Service Invocation Listeners

Service Invocation Listeners are components that listen to the specific low level signals related to the invocation or stopping of a service. They can either be service specific or generic listeners. A given service may find useful to configure and use any of this existing generic listeners. Service Invocation Listeners send an event to the PBMS if certain conditions are met in that signal. The PBMS may perform control operations on the service (e.g. start or stop its execution), based on the information received in the event and the service execution policies that have been defined.

In general, a listener can be implemented to monitor the appropriate devices in different ways. Nevertheless, the active layer provides a simple and flexible platform for this kind of jobs. Thus, the listeners may be implemented as an active service that uses the different brokers to perform its monitoring operations.

A.3 Brief Rationale for selecting XML as the policy syntax

Various specifications for data formats exists, both in their syntax and semantic. To represent policies accurately and effectively, the syntax, semantic (metadata) and API for implementation have to be considered. This sub-section focuses on the syntax representation of policy, to which, widely used mark-up language will be used.

Among all mark-up languages, eXtensible Mark-up Language (XML) is becoming increasingly adopted as a common syntax for expressing structure in data. XML is particularly suitable as a data representation mechanism for use in heterogeneous environments.

XML is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

Because XML is based upon an open industry standard, implementations of XML parsers exist for many platforms, and in many programming languages. Implementations are available on Unix and in C++ and Java. In addition, some parser implementations allow access to XML element tree from various different scripting languages and environments. XML's tree structures allow for context sensitivity and a more direct mapping to the object model than if flat parameter lists or logic expressions are used.

XML's significant advantages make it a good choice for policy representation for network management . In policy-based network management environments, the XML representation of policies could be used to transfer policies between co-operating management platforms. These management platforms need not necessarily be running the same operating environments, but interoperability is enabled because of a common understanding of the XML representation of policy.

Furthermore, a wide variety of XML supporting tools exist to allow a fast policy-based management (PBM) development.

But XML does not provide any semantics for its document, and this is completely up to the users, which means that the semantics for that XML file have to be developed.

A3.1.1 XML Schema

A document written in XML has to conform to some rules to be understandable and executable. These rules can basically be divided into two categories:

- Document Type Definition (DTD)
- XML Schema.

Schema is already a standard set by World Wide Web Consortium (W3C), and is more powerful and flexible. As such, Schema mode is suggested to be used in CONTEXT. Within the Schema mode, there are still various specifications, of which Resource Description Framework (RDF) and W3C Schema are most famous.

A3.1.2 RDF

RDF provides a common basis for expressing semantics in XML documents. It is a functional layer above XML. Applications, which allow programs to combine data logically, will be built using RDF (and therefore XML) and this will enhance the modularity and extensibility of the policy. One of the requirements of PBM in CONTEXT is to allow huge amounts of policies to be stored in databases and existing applications to be put on the network, not just for user browsing, but also for machine understanding, i.e., searching, reasoning and analysing. This will need the help of metadata. RDF allows metadata applications to be combined, and to operate in a common way as the semantics that they share.

The RDF specifications also provide a lightweight ontology system to support the exchange of knowledge on the network, which is also a requirement for active networks. We intend to look for RDF specifications specific to PBM, otherwise, a novel one would be developed.

Almost all the tools supporting XML also support RDF, so all these tools will be discussed later.

A3.1.3 W3C Schema

XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. Numerous information about W3C Schema is available on WWW.

A3.1.4 XML API

After the syntax and semantic of the policies have been determined, the next crucial issue to address is the implementation of these policies. Applications need to be developed in order to take an XML document and make its structure and content available to component that needs it, e.g., a PEP or even a node operating system (NodeOS) in the CONTEXT architecture. XML APIs are required for this task.

In general, there are two major types of XML APIs, i.e.,

- tree-based APIs
- event-based APIs.

A tree-based API compiles an XML document into an internal tree structure, and then allows an application to navigate that tree. A common example is the Document Object Model (DOM).

On the other hand, an event-based API reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events - much like handling events in a graphical user interface. The typical one of this type is Simple API for XML (SAX).

While DOM APIs are useful for a wide range of applications, they often put a great strain on system resources, especially if the document is large. Under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem. Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

SAX API provides a simpler, lower-level access to an XML document. Documents much larger than available system memory can be parsed and one can construct specific data structures using callback event handlers.

A3.1.5 Tools For XML Processing

A3.1.5.1 XML Browser

When working with XML, we will need to view XML documents. The tool for this purpose is usually called XML browser. XML browsers are generally driven by style sheets, and are tree-based. They can be generic ones such as IE5 and Netscape, or application specific ones. XML editors, as shown below, usually also have this functionality.

A3.1.5.2 XML Editor and XML Spy

For input of XML document, the ordinary text editor works. But if special XML editors are used, less mistakes will be made. Many XML editors are sensitive to Schemas and/or DTDs and so can enable user to easily produce well-formed and valid XML documents. Cooktop is a good option, but XML Spy [69] is more powerful. More XML tools can be found on XMLSOFTWARE [70], which aims to provide well organised information, resources, especially software tools on XML.

XML Spy is the first true Integrated Development Environment for XML that includes all major aspects of XML in one powerful and easy-to-use product, which can be downloaded free of charge. XML Spy is centered around a professional validating XML editor that provides five advanced views on your documents: an Enhanced Grid View for structured editing, a Database/Table view that shows repeated elements in a tabular fashion, a Text View with syntax-coloring for low-level work, a graphical XML Schema design view, and an integrated Browser View that supports both CSS and XSL style-sheets.

A3.1.5.3 XML Parser and Sun JAXP

A parser or processor takes an XML document and makes its structure and content available to an application, often via a standard interface like SAX or DOM. Basically, XML Parser [71] can be divided into two categories: application/product specific XML parser, such as Oracle XML parser for C/Java, and generic XML parser.

XML parser also language-oriented. So there are XML parser for C, XML parser for Java, XML parser for COBOL, XML parser for PL/SQL, etc.

Generic XML parsers can be used of which Sun Java API for XML Processing (JAXP) [72] is highly appreciated. JAXP enables applications to parse and transform XML documents using a pure Java API that is independent of a particular XML processor implementation. Depending on the needs of the application, developers have the flexibility to swap between XML processors (such as high performance vs. memory conservative parsers) without making application code changes. Thus, application and tools developers can rapidly and easily XML-enable their Java applications.

The reference implementation uses Crimson [73], which was derived from the Java Project X parser from Sun, as its default XML parser and Xalan [74] as its default XSLT engine. Therefore, JAXP 1.1 also supports DOM Level 2 and SAX version 2.0. However, the pluggable architecture of JAXP allows any XML conformant implementations to be used.

A.4 Test Policies

A4.1 ConflictTestPolicy1

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
    <Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Policy_Set_Id>ConflictTest</Policy_Set_Id>
        <Policy_Group_Id/>conflict</Policy_Group_Id>
        <Policy_Id>Policy1</Policy_Id>
        <Policy_Aim>new</Policy_Aim>
        <IsAtomic>true</IsAtomic>
        <Policy_Sequence_Position>1</Policy_Sequence_Position>
        <Validity_Period>
            <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
        </Validity_Period>
        <Condition>
            <Condition_Object>
                <Event>
                    <Event_Id>ev1</Event_Id>
                    <Event_Type>demoEv1</Event_Type>
                    <Event_Variable>
                        <Variable_Type>demoEvVar1</Variable_Type>
                        <Syntax>String</Syntax>
                    </Event_Variable>
                    <Monitoring_Component>
                        <Component_Name>demoPEP</Component_Name>
                    </Monitoring_Component>
                </Event>
            </Condition_Object>
            <Condition_Requirement>
                <Requirement_Id>Requirement2</Requirement_Id>
                <Requirement_Type>Match_Value</Requirement_Type>
                <Target_Condition_Object>
                    <Event>
                        <Event_Id>ev1</Event_Id>
                        <Event_Variable_Type>demoEvVar1</Event_Variable_Type>
                    </Event>
                </Target_Condition_Object>
                <Requirement_Parameter>
                    <Name>Match_Value</Name>
                    <Syntax>String</Syntax>
```

```
                <Value>Hello</Value>
            </Requirement_Parameter>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
    </Condition>
    <Enforcement_Sequence>Action1</Enforcement_Sequence>
    <Action>
        <Action_Id>Action1</Action_Id>
        <Action_Type>openApplet</Action_Type>
        <Action_Parameter>
            <Name>arg</Name>
            <Syntax>String[]</Syntax>
            <Value>blue</Value>
        </Action_Parameter>
        <Enforcer_Component>
            <Component_Name>DemoPEP</Component_Name>
        </Enforcer_Component>
        <Enforcement_TimeOut>1min</Enforcement_TimeOut>
        <Success_Output_Parameters>
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
</Policy>
```

A4.2 ConflictTestPolicy2

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
    <Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Policy_Set_Id>ConflictTest</Policy_Set_Id>
        <Policy_Group_Id/>conflict</Policy_Group_Id>
        <Policy_Id>Policy1</Policy_Id>
        <Policy_Aim>new</Policy_Aim>
        <IsAtomic>true</IsAtomic>
        <Policy_Sequence_Position>1</Policy_Sequence_Position>
        <Validity_Period>
            <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
        </Validity_Period>
        <Condition>
            <Condition_Object>
                <Event>
```

```
            <Event_Id>ev1</Event_Id>
            <Event_Type>demoEv1</Event_Type>
            <Event_Variable>
                <Variable_Type>demoEvVar1</Variable_Type>
                <Syntax>String</Syntax>
            </Event_Variable>
            <Monitoring_Component>
                <Component_Name>demoPEP</Component_Name>
            </Monitoring_Component>
        </Event>
    </Condition_Object>
    <Condition_Requirement>
        <Requirement_Id>Requirement2</Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Target_Condition_Object>
            <Event>
                <Event_Id>ev1</Event_Id>
                <Event_Variable_Type>demoEvVar1</Event_Variable_Type>
            </Event>
        </Target_Condition_Object>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
            <Value>Hello</Value>
        </Requirement_Parameter>
        <Req_Evaluator_Component>
            <Component_Name>demoPEP</Component_Name>
        </Req_Evaluator_Component>
    </Condition_Requirement>
</Condition>
<Enforcement_Sequence>Action1</Enforcement_Sequence>
<Action>
    <Action_Id>Action1</Action_Id>
    <Action_Type>openApplet</Action_Type>
    <Action_Parameter>
        <Name>arg</Name>
        <Syntax>String[]</Syntax>
        <Value>Black</Value>
    </Action_Parameter>
    <Enforcer_Component>
        <Component_Name>DemoPEP</Component_Name>
    </Enforcer_Component>
    <Enforcement_TimeOut>1min</Enforcement_TimeOut>
    <Success_Output_Parameters>
```

```
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
    </Policy>
```

A4.3 ConditionTestPolicy

The ConditionTestPolicy is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id/>
    <Policy_Group_Id/>
    <Policy_Id>Policy1</Policy_Id>
    <Policy_Aim>new</Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Policy_Sequence_Position>1</Policy_Sequence_Position>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id>var1</Variable_Id>
                <Variable_Type>demoVar1</Variable_Type>
                <Syntax>Double</Syntax>
                <Monitoring_Component>
                    <Component_Name>demoCE</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id>var2</Variable_Id>
                <Variable_Type>demoVar2</Variable_Type>
                <Syntax>Double</Syntax>
                <Monitoring_Component>
                    <Component_Name>demoCE</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Object>
            <Simple_Variable>
```

```
            <Variable_Id>var3</Variable_Id>
            <Variable_Type>demoVar3</Variable_Type>
            <Syntax>Double</Syntax>
            <Monitoring_Component>
                <Component_Name>demoCE</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Object>
        <Aggregated_Variable>
            <Variable_Id>AVar1</Variable_Id>
            <Syntax>Double</Syntax>
            <Operation>(var1)+(var2)+(var3)</Operation>
        </Aggregated_Variable>
    </Condition_Object>
    <Condition_Object>
        <Event>
            <Event_Id>ev1</Event_Id>
            <Event_Type>demoEv1</Event_Type>
            <Event_Variable>
                <Variable_Type>demoEvVar1</Variable_Type>
                <Syntax>String</Syntax>
            </Event_Variable>
            <Monitoring_Component>
                <Component_Name>demoPEP</Component_Name>
            </Monitoring_Component>
        </Event>
    </Condition_Object>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id>var4</Variable_Id>
            <Variable_Type>demoVar4</Variable_Type>
            <Syntax>Double</Syntax>
            <Monitoring_Component>
                <Component_Name>demoCE</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id>var5</Variable_Id>
            <Variable_Type>demoVar5</Variable_Type>
            <Syntax>Double</Syntax>
            <Monitoring_Component>
```

```
            <Component_Name>demoPEP</Component_Name>
        </Monitoring_Component>
    </Simple_Variable>
</Condition_Object>
<Condition_Requirement>
    <Requirement_Id>Requirement1</Requirement_Id>
    <Requirement_Type>Out_Margins</Requirement_Type>
    <Target_Condition_Object>
        <Aggregated_Variable_Id>AVar1</Aggregated_Variable_Id>
    </Target_Condition_Object>
    <Requirement_Parameter>
        <Name>High_Margin</Name>
        <Syntax>Double</Syntax>
        <Value>10</Value>
    </Requirement_Parameter>
    <Requirement_Parameter>
        <Name>Low_Margin</Name>
        <Syntax>Double</Syntax>
        <Value>5</Value>
    </Requirement_Parameter>
    <Req_Evaluator_Component>
        <Component_Name>AVCE</Component_Name>
    </Req_Evaluator_Component>
</Condition_Requirement>
<Condition_Requirement>
    <Requirement_Id>Requirement2</Requirement_Id>
    <Requirement_Type>Match_Value</Requirement_Type>
    <Target_Condition_Object>
        <Event>
            <Event_Id>ev1</Event_Id>
            <Event_Variable_Type>demoEvVar1</Event_Variable_Type>
        </Event>
    </Target_Condition_Object>
    <Requirement_Parameter>
        <Name>Match_Value</Name>
        <Syntax>String</Syntax>
        <Value>Hello</Value>
    </Requirement_Parameter>
    <Req_Evaluator_Component>
        <Component_Name>demoPEP</Component_Name>
    </Req_Evaluator_Component>
</Condition_Requirement>
<Condition_Requirement>
    <Requirement_Id>Requirement3</Requirement_Id>
```

```
            <Requirement_Type>Max_Threshold</Requirement_Type>
            <Target_Condition_Object>
                <Simple_Variable_Id>var4</Simple_Variable_Id>
            </Target_Condition_Object>
            <Requirement_Parameter>
                <Name>Threshold</Name>
                <Syntax>Double</Syntax>
                <Value>100</Value>
            </Requirement_Parameter>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Requirement>
            <Requirement_Id>Requirement4</Requirement_Id>
            <Requirement_Type>Min_Threshold</Requirement_Type>
            <Target_Condition_Object>
                <Simple_Variable_Id>var5</Simple_Variable_Id>
            </Target_Condition_Object>
            <Requirement_Parameter>
                <Name>Threshold</Name>
                <Syntax>Double</Syntax>
                <Value>100</Value>
            </Requirement_Parameter>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
            <Condition_Evaluation_Method>(Requirement1 AND Requirement2) OR (Requirement3 AND
Requirement4)</Condition_Evaluation_Method>
            <Condition_Evaluation_Stop>3</Condition_Evaluation_Stop>
        </Condition_Evaluation_Parameters>
    </Condition>
    <Enforcement_Sequence>Action1</Enforcement_Sequence>
    <Action>
        <Action_Id>Action1</Action_Id>
        <Action_Type>openApplet</Action_Type>
        <Action_Parameter>
            <Name>arg</Name>
            <Syntax>String[]</Syntax>
            <Value>blue</Value>
        </Action_Parameter>
        <Enforcer_Component>
```

```
            <Component_Name>DemoPEP</Component_Name>
        </Enforcer_Component>
        <Enforcement_TimeOut>1min</Enforcement_TimeOut>
        <Success_Output_Parameters>
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
</Policy>
```

A4.4 ActionTestPolicy1

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id/>
    <Policy_Group_Id/>
    <Policy_Id>policy2</Policy_Id>
    <Policy_Aim>new</Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Policy_Sequence_Position>1</Policy_Sequence_Position>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id>var1</Variable_Id>
                <Variable_Type>demoVar1</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
                    <Component_Name>demoPEP</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Requirement>
            <Requirement_Id>Requirement1</Requirement_Id>
            <Requirement_Type>Match_Value</Requirement_Type>
            <Target_Condition_Object>
                <Simple_Variable_Id>var1</Simple_Variable_Id>
            </Target_Condition_Object>
            <Requirement_Parameter>
```

```
                <Name>Match_Value</Name>
                <Syntax>String</Syntax>
                <Value>ANY</Value>
            </Requirement_Parameter>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
            <Condition_Evaluation_Method>Requirement1</Condition_Evaluation_Method>
            <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
        </Condition_Evaluation_Parameters>
    </Condition>
    <Enforcement_Sequence>Action1</Enforcement_Sequence>
    <Action>
        <Action_Id>Action1</Action_Id>
        <Action_Type>openApplet</Action_Type>
        <Action_Parameter>
            <Name>arg</Name>
            <Syntax>String[]</Syntax>
            <Condition_Object>
                <Simple_Variable_Id>var1</Simple_Variable_Id>
            </Condition_Object>
        </Action_Parameter>
        <Enforcer_Component>
            <Component_Name>DemoPEP</Component_Name>
        </Enforcer_Component>
        <Enforcement_TimeOut>1min</Enforcement_TimeOut>
        <Success_Output_Parameters>
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
</Policy>
```

A4.5 LocalVariableTest1 policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
```

```
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id/>
    <Policy_Group_Id/>
    <Policy_Id>localVarPolicy1</Policy_Id>
    <Policy_Aim>new</Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Policy_Sequence_Position>1</Policy_Sequence_Position>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id>var1</Variable_Id>
                <Variable_Type>varType</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
                    <Component_Name>demoPEP</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Requirement>
            <Requirement_Id>Requirement1</Requirement_Id>
            <Requirement_Type>Match_Value</Requirement_Type>
            <Target_Condition_Object>
                <Simple_Variable_Id>var1</Simple_Variable_Id>
            </Target_Condition_Object>
            <Requirement_Parameter>
                <Name>Match_Value</Name>
                <Syntax>String</Syntax>
                <Value>hello world</Value>
            </Requirement_Parameter>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
            <Condition_Evaluation_Method>Requirement1</Condition_Evaluation_Method>
            <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
        </Condition_Evaluation_Parameters>
    </Condition>
    <Enforcement_Sequence>Action11</Enforcement_Sequence>
    <Action>
```

```
            <Action_Id>Action11</Action_Id>
            <Action_Type>setLocalVariable</Action_Type>
            <Action_Parameter>
                <Name>Group</Name>
                <Syntax>String</Syntax>
                <Value>LVGroupExample</Value>
            </Action_Parameter>
            <Action_Parameter>
                <Name>Name</Name>
                <Syntax>String</Syntax>
                <Value>lvar1</Value>
            </Action_Parameter>
            <Action_Parameter>
                <Name>Syntax</Name>
                <Syntax>String</Syntax>
                <Value>String</Value>
            </Action_Parameter>
            <Action_Parameter>
                <Name>Value</Name>
                <Syntax>String</Syntax>
                <Value>blue</Value>
            </Action_Parameter>
            <Enforcer_Component>
                <Component_Name>demoPEP</Component_Name>
            </Enforcer_Component>
            <Enforcement_TimeOut>1min</Enforcement_TimeOut>
            <Success_Output_Parameters>
                <Action_Result_Value>success</Action_Result_Value>
            </Success_Output_Parameters>
        </Action>
    </Policy>
```

A4.6 LocalVariableTest2 policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
        <Policy_Set_Id/>
        <Policy_Group_Id/>
        <Policy_Id>localVarPolicy2</Policy_Id>
```

```
<Policy_Aim>new</Policy_Aim>
<IsAtomic>true</IsAtomic>
<Policy_Sequence_Position>2</Policy_Sequence_Position>
<Validity_Period>
    <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
</Validity_Period>
<Condition>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id>lvar1</Variable_Id>
            <Variable_Type>LocalVariable</Variable_Type>
            <Syntax>String</Syntax>
            <Monitoring_Component>
                <Component_Name>demoPEP</Component_Name>
            </Monitoring_Component>
            <Monitoring_Parameter>
                <Name>Group</Name>
                <Syntax>String</Syntax>
                <Value>LVGroupExample</Value>
            </Monitoring_Parameter>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Requirement>
        <Requirement_Id>Requirement1</Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Target_Condition_Object>
            <Simple_Variable_Id>lvar1</Simple_Variable_Id>
        </Target_Condition_Object>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
            <Value>blue</Value>
        </Requirement_Parameter>
        <Req_Evaluator_Component>
            <Component_Name>demoPEP</Component_Name>
        </Req_Evaluator_Component>
    </Condition_Requirement>
    <Condition_Evaluation_Parameters>
        <Condition_Evaluation_Method>Requirement1</Condition_Evaluation_Method>
        <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
    </Condition_Evaluation_Parameters>
</Condition>
<Enforcement_Sequence>Action21</Enforcement_Sequence>
<Action>
```

```
            <Action_Id>Action21</Action_Id>
            <Action_Type>openApplet</Action_Type>
            <Action_Parameter>
                <Name>arg</Name>
                <Syntax>String[]</Syntax>
                <Value>blue</Value>
            </Action_Parameter>
            <Enforcer_Component>
                <Component_Name>demoPEP</Component_Name>
            </Enforcer_Component>
            <Enforcement_TimeOut>1min</Enforcement_TimeOut>
            <Success_Output_Parameters>
                <Action_Result_Value>success</Action_Result_Value>
            </Success_Output_Parameters>
        </Action>
    </Policy>
```

A4.7 Policy111

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Policy_Set_Id>PSetTest1</Policy_Set_Id>
        <Policy_Group_Id>PGroupTest11</Policy_Group_Id>
        <Policy_Id>policy111</Policy_Id>
        <Policy_Aim>new</Policy_Aim>
        <IsAtomic>true</IsAtomic>
        <Policy_Sequence_Position>1</Policy_Sequence_Position>
        <Validity_Period>
            <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
        </Validity_Period>
        <Condition>
            <Condition_Object>
                <Simple_Variable>
                    <Variable_Id>var1</Variable_Id>
                    <Variable_Type>demoVar1</Variable_Type>
                    <Syntax>String</Syntax>
                    <Monitoring_Component>
```

```
                    <Component_Name>demoPEP</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Requirement>
            <Requirement_Id>Requirement11</Requirement_Id>
            <Requirement_Type>Match_Value</Requirement_Type>
            <Requirement_Parameter>
                <Name>Match_Value</Name>
                <Syntax>String</Syntax>
                <Value>hello</Value>
            </Requirement_Parameter>
            <Target_Condition_Object>
                <Simple_Variable_Id>var1</Simple_Variable_Id>
            </Target_Condition_Object>
            <Req_Evaluator_Component>
                <Component_Name>demoPEP</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
            <Condition_Evaluation_Method>Requirement11</Condition_Evaluation_Method>
            <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
        </Condition_Evaluation_Parameters>
    </Condition>
    <Enforcement_Sequence>Action11</Enforcement_Sequence>
    <Action>
        <Action_Id>Action11</Action_Id>
        <Action_Type>openApplet</Action_Type>
        <Action_Parameter>
            <Name>arg</Name>
            <Syntax>String[]</Syntax>
            <Value>blue</Value>
        </Action_Parameter>
        <Enforcer_Component>
            <Component_Name>demoPEP</Component_Name>
        </Enforcer_Component>
        <Enforcement_TimeOut>1min</Enforcement_TimeOut>
        <Success_Output_Parameters>
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
</Policy>
```

A4.8 FollowMe Policies

A4.8.1 The Service Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
        <policy_Set_id>FollowMe</policy_Set_id>
        <policy_Group_id>Service</policy_Group_id>
        <policyId>detectUser</policyId>
        <Policy_Aim>detect Service instantiation</Policy_Aim>
        <isAtomic>true</isAtomic>
        <Validity_Period>
                <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>

        </Validity_Period>
        <Condition>
                <Condition_Object>
                        <Event>
                                <Event_Id>Event1</Event_Id>
                                <Event_Type> new_user_in_coverage_area </Event_Type>
                                <Event_Variable>
                                        <Variable_Type>new_user_MAC_address </Variable_Type>
                                        <Syntax>String</Syntax>
                                </Event_Variable>
                                <Monitoring_Component>
                                        <Component_Name>WLANBroker</Component_Name>
                                        <Component_Location>isobitis.ee.ucl.ac.uk
                                        </Component_Location>
                                </Monitoring_Component>

                        </Event>
                </Condition_Object>
                <Condition_Requirement>
                        <Requirement_Id>new_user</Requirement_Id>
                        <Requirement_Type>Match_Value</Requirement_Type>
                        <Target_Condition_Object>
                                <Event>
                                        <Event_Id>Event1</Event_Id>
                                        <Event_Variable_Type>new_user_in_coverage_area
                                        </Event_Variable_Type>
                                </Event>
                        </Target_Condition_Object>
                        <Requirement_Parameter>
                                <Name>Event1</Name>
                                <Syntax>String</Syntax>
                                <Value>00-0E-35-37-EB-B4</Value>
                        </Requirement_Parameter>
                        <Req_Evaluator_Component>
                                <Component_Name>WLANBroker</Component_Name>
                                <Component_Location> isobitis.ee.ucl.ac.uk
                                 </Component_Location>
                        </Req_Evaluator_Component>
```

```
            </Condition_Requirement>
            <Condition_Evaluation_Parameters>
                <Condition_Evaluation_Method>String</Condition_Evaluation_Method>

        <Condition_Evaluation_Periodicity>String</Condition_Evaluation_Periodicity>
                <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
            </Condition_Evaluation_Parameters>
        </Condition>
        <PolicyVariable>
            <ContextVariable>
                <ContextEntity>
                    <Person>
                        <name>Nikos Vardalachos</name>
                        <username>nvardala</username>
                        <password>nikos</password>
                    </Person>
                    <NetMngtStation>
                        <name>Nikos Vardalachos</name>
                        <key>HospitalCustNet</key>
                        <ofType>wlan</ ofType >
                    </ NetMngtStation >

                </ContextEntity>
                <Activity>InRange</Activity>
            </ContextVariable>
        </PolicyVariable>

        <Action>
            <Action_Id>Event1</Action_Id>
            <Action_Type>setLocalVariable</Action_Type>
            <Action_Parameter>
                <Name>setServiceActive</Name>
                <Syntax>String[]</Syntax>
                <Value>true</Value>
            </Action_Parameter>
            <Enforcer_Component>
                <Component_Name>PolicyManager</Component_Name>
            </Enforcer_Component>
            <Success_Output_Parameters>
                <Action_Result_Value/>
                <Output_Parameter>
                    <Name>serviceResult</Name>
                    <Syntax>String</Syntax>
                    <Value>success</Value>
                </Output_Parameter>
            </Success_Output_Parameters>
        </Action>
</policy>
```

A4.8.2 The Tunnel Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id>FollowMe</Policy_Set_Id>
    <Policy_Group_Id>Tunnel</Policy_Group_Id>
    <Policy_Id>setup</Policy_Id>
    <Policy_Aim> setupTunnel </Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id> setServiceActive </Variable_Id>
                <Variable_Type>serviceActive</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
                    <Component_Name>Policy Manager</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Requirement>
            <Requirement_Id> setServiceActive </Requirement_Id>
            <Requirement_Type>Match_Value</Requirement_Type>
            <Requirement_Parameter>
                <Name>Match_Value</Name>
                <Syntax>String</Syntax>
                <Value> true</Value>
            </Requirement_Parameter>
            <Target_Condition_Object>
                <Simple_Variable_Id> setServiceActive </Simple_Variable_Id>
            </Target_Condition_Object>
            <Req_Evaluator_Component>
                <Component_Name>PolicyManager</Component_Name>
            </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
            <Condition_Evaluation_Method>serviceActive </Condition_Evaluation_Method>
```

```
        <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
    </Condition_Evaluation_Parameters>
</Condition>
        <PolicyVariable>
        <ContextVariable>
                <ContextEntity>
                        <Person>
                                <name>Nikos Vardalachos</name>
                                <username>nvardala</username>
                                <password>nikos</password>
                        </Person>
                        <Device>
                                <NetworkElement>
                                        <name>vpn1</name>
                                        <type>IP-GRE</type>
                                        <netId>192.689.1.2</ netId >
                                </ NetworkElement>
                        </ Device >

                </ContextEntity>
                <Activity>InRange</Activity>
        </ContextVariable>
 </PolicyVariable>

<Action>
    <Action_Id>setUpTunnels</Action_Id>
    <Action_Type>tunnelSet</Action_Type>
    <Action_Parameter>
        <Name>tunnelSet</Name>
        <Syntax>String[]</Syntax>
        <Value>true</Value>
    </Action_Parameter>
    <Enforcer_Component>
        <Component_Name>WLAN Broker</Component_Name>
    </Enforcer_Component>
    <Enforcement_TimeOut>1min</Enforcement_TimeOut>
    <Success_Output_Parameters>
        <Action_Result_Value>success</Action_Result_Value>
    </Success_Output_Parameters>
</Action>
<Action>
  <Action_Id>tunnelSetStatus</Action_Id>
  <Action_Type>setLocalVariable</Action_Type>
        <Action_Parameter>
                <Name>tunnelSetStatus</Name>
                <Syntax>String[]</Syntax>
                <Value>true</Value>
        </Action_Parameter>
        <Enforcer_Component>
                <Component_Name>PolicyManager</Component_Name>
```

```
            </Enforcer_Component>
            <Success_Output_Parameters>
                <Action_Result_Value/>
                <Output_Parameter>
                    <Name>tunnelResult</Name>
                    <Syntax>String</Syntax>
                    <Value>success</Value>
                </Output_Parameter>
            </Success_Output_Parameters>
        </Action>

    </Policy>
```

A4.8.3 The Firewall Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id>FollowMe</Policy_Set_Id>
    <Policy_Group_Id>Firewall</Policy_Group_Id>
    <Policy_Id>setupFirewall</Policy_Id>
    <Policy_Aim> setFirewall </Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id> setServiceActive </Variable_Id>
                <Variable_Type>serviceActive</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
                    <Component_Name>Policy Manager</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id> tunnelSetstatus </Variable_Id>
                <Variable_Type>tunnelSet</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
```

```
                <Component_Name>WLANBroker</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>

    <Condition_Requirement>
        <Requirement_Id> setServiceActive </Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
            <Value> true</Value>
        </Requirement_Parameter>
        <Target_Condition_Object>
            <Simple_Variable_Id> setServiceActive </Simple_Variable_Id>
        </Target_Condition_Object>
        <Target_Condition_Object>
            <Simple_Variable_Id> tunnelSetStatus </Simple_Variable_Id>
        </Target_Condition_Object>

        <Req_Evaluator_Component>
            <Component_Name>PolicyManager</Component_Name>
        </Req_Evaluator_Component>
    </Condition_Requirement>
    <Condition_Evaluation_Parameters>
        <Condition_Evaluation_Method>(setServiceActive) AND (tunnelSetStatus)
        </Condition_Evaluation_Method>
        <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
    </Condition_Evaluation_Parameters>
</Condition>
        <PolicyVariable>
        <ContextVariable>
                <ContextEntity>

                        <Device>
                                <NetworkElement>
                                        <name>fwrule</name>
                                        <type>ipchains add</type>

                                </ NetworkElement>
                        </ Device >

                </ContextEntity>
                <Activity>InRange</Activity>
        </ContextVariable>
 </PolicyVariable>
```

```
    <Action>
        <Action_Id>setUpFirewall</Action_Id>
        <Action_Type>firewallISet</Action_Type>
        <Action_Parameter>
            <Name>firewallSetStatus</Name>
            <Syntax>String[]</Syntax>
            <Value>true</Value>
        </Action_Parameter>
        <Enforcer_Component>
            <Component_Name>WLAN Broker</Component_Name>
        </Enforcer_Component>
        <Enforcement_TimeOut>1min</Enforcement_TimeOut>
        <Success_Output_Parameters>
            <Action_Result_Value>success</Action_Result_Value>
        </Success_Output_Parameters>
    </Action>
    <Action>
      <Action_Id>firewallSetStatus</Action_Id>
      <Action_Type>setLocalVariable</Action_Type>
            <Action_Parameter>
                    <Name>firewallSetStatus</Name>
                    <Syntax>String[]</Syntax>
                    <Value>true</Value>
            </Action_Parameter>
            <Enforcer_Component>
                    <Component_Name>PolicyManager</Component_Name>
            </Enforcer_Component>
            <Success_Output_Parameters>
                    <Action_Result_Value/>
                    <Output_Parameter>
                            <Name>firewallResult</Name>
                            <Syntax>String</Syntax>
                            <Value>success</Value>
                    </Output_Parameter>
            </Success_Output_Parameters>
      </Action>

</Policy>
```

A4.8.4 The Routing Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id>FollowMe</Policy_Set_Id>
    <Policy_Group_Id>Routing</Policy_Group_Id>
```

```
<Policy_Id>setupRoutes</Policy_Id>
<Policy_Aim> setRoutes </Policy_Aim>
<IsAtomic>true</IsAtomic>
<Validity_Period>
    <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
</Validity_Period>
<Condition>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id> setServiceActive </Variable_Id>
            <Variable_Type>serviceActive</Variable_Type>
            <Syntax>String</Syntax>
            <Monitoring_Component>
                <Component_Name>Policy Manager</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id> tunnelSetstatus </Variable_Id>
            <Variable_Type>tunnelSet</Variable_Type>
            <Syntax>String</Syntax>
            <Monitoring_Component>
                <Component_Name>WLANBroker</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Object>
        <Simple_Variable>
            <Variable_Id> firewallSetstatus </Variable_Id>
            <Variable_Type>firewallSet</Variable_Type>
            <Syntax>String</Syntax>
            <Monitoring_Component>
                <Component_Name>WLANBroker</Component_Name>
            </Monitoring_Component>
        </Simple_Variable>
    </Condition_Object>
    <Condition_Requirement>
        <Requirement_Id> setServiceActive </Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
```

```
            <Value> true</Value>
        </Requirement_Parameter>
        <Condition_Requirement>
        <Requirement_Id> tunnelSetStatus </Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
            <Value> true</Value>
        </Requirement_Parameter>
        <Condition_Requirement>
        <Requirement_Id> firewallSetStatus </Requirement_Id>
        <Requirement_Type>Match_Value</Requirement_Type>
        <Requirement_Parameter>
            <Name>Match_Value</Name>
            <Syntax>String</Syntax>
            <Value> true</Value>
        </Requirement_Parameter>

        <Target_Condition_Object>
            <Simple_Variable_Id> setServiceActive </Simple_Variable_Id>
        </Target_Condition_Object>
        <Target_Condition_Object>
            <Simple_Variable_Id> tunnelSetStatus </Simple_Variable_Id>
        </Target_Condition_Object>
        <Target_Condition_Object>
            <Simple_Variable_Id> firewallSetStatus </Simple_Variable_Id>
        </Target_Condition_Object>

        <Req_Evaluator_Component>
            <Component_Name>PolicyManager</Component_Name>
        </Req_Evaluator_Component>
    </Condition_Requirement>
    <Condition_Evaluation_Parameters>
        <Condition_Evaluation_Method>(setServiceActive) AND (tunnelSetStatus) AND (firewallSetStatus)
</Condition_Evaluation_Method>
        <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
    </Condition_Evaluation_Parameters>
</Condition>
        <PolicyVariable>
        <ContextVariable>
            <ContextEntity>

                <Device>
```

```
                                        <NetworkElement>
                                                <name>route</name>
                                                <type>ip-route add</type>

                                        </ NetworkElement>
                                    </ Device >

                            </ContextEntity>
                            <Activity>InRange</Activity>
                        </ContextVariable>
                </PolicyVariable>

                <Action>
                    <Action_Id>setUproutes</Action_Id>
                    <Action_Type>routeSet</Action_Type>
                    <Action_Parameter>
                        <Name>routeSetStatus</Name>
                        <Syntax>String[]</Syntax>
                        <Value>true</Value>
                    </Action_Parameter>
                    <Enforcer_Component>
                        <Component_Name>WLAN Broker</Component_Name>
                    </Enforcer_Component>
                    <Enforcement_TimeOut>1min</Enforcement_TimeOut>
                    <Success_Output_Parameters>
                        <Action_Result_Value>success</Action_Result_Value>
                    </Success_Output_Parameters>
                </Action>
            </Policy>
```

A4.9 EmergencySupport Policies

A4.9.1 The Service Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
        <policy_Set_id>EmergencySupport</policy_Set_id>
        <policy_Group_id>Service</policy_Group_id>
        <policyId>emergencyOn</policyId>
        <Policy_Aim>detect Service instantiation</Policy_Aim>
        <isAtomic>true</isAtomic>
        <Validity_Period>
                <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>

        </Validity_Period>
        <Condition>
```

```
        <Condition_Object>
                <Event>
                        <Event_Id>Event1</Event_Id>
                        <Event_Type> sipCall</Event_Type>
                        <Event_Variable>
                                <Variable_Type>new_sip_call </Variable_Type>
                                <Syntax>String</Syntax>
                        </Event_Variable>
                        <Monitoring_Component>
                                <Component_Name>SIPBroker</Component_Name>
                                <Component_Location>isobitis.ee.ucl.ac.uk
                                </Component_Location>
                        </Monitoring_Component>

                </Event>
        </Condition_Object>
        <Condition_Requirement>
                <Requirement_Id>new_sip</Requirement_Id>
                <Requirement_Type>Max_Threshold</Requirement_Type>
                <Target_Condition_Object>
                        <Event>
                                <Event_Id>Event1</Event_Id>
                                <Event_Variable_Type>new_sip_call
                                </Event_Variable_Type>
                        </Event>
                </Target_Condition_Object>
                <Requirement_Parameter>
                        <Name>Event1</Name>
                        <Syntax>String</Syntax>
                        <Value>3</Value>
                </Requirement_Parameter>
                <Req_Evaluator_Component>
                        <Component_Name>SIPBroker</Component_Name>
                        <Component_Location> isobitis.ee.ucl.ac.uk
                         </Component_Location>
                </Req_Evaluator_Component>
        </Condition_Requirement>
        <Condition_Evaluation_Parameters>
                <Condition_Evaluation_Method>String</Condition_Evaluation_Method>

<Condition_Evaluation_Periodicity>String</Condition_Evaluation_Periodicity>
                <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
        </Condition_Evaluation_Parameters>
</Condition>


<Action>
        <Action_Id>Event1</Action_Id>
        <Action_Type>setLocalVariable</Action_Type>
        <Action_Parameter>
                <Name>setEmergencyOn</Name>
                <Syntax>String[]</Syntax>
                <Value>true</Value>
        </Action_Parameter>
        <Enforcer_Component>
```

```
                <Component_Name>PolicyManager</Component_Name>
            </Enforcer_Component>
            <Success_Output_Parameters>
                <Action_Result_Value/>
                <Output_Parameter>
                    <Name>serviceResult</Name>
                    <Syntax>String</Syntax>
                    <Value>success</Value>
                </Output_Parameter>
            </Success_Output_Parameters>
        </Action>
</policy>
```

A4.9.2 The sipAccess Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by NIKOS -->
<policy                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Nikos\current\Phd - final\testPoli.xsd">
    <Policy_Set_Id>EmergencySupport</Policy_Set_Id>
    <Policy_Group_Id>sipControl</Policy_Group_Id>
    <Policy_Id>setup</Policy_Id>
    <Policy_Aim> setup access control </Policy_Aim>
    <IsAtomic>true</IsAtomic>
    <Validity_Period>
        <Time_Period>18/06/04-9:00;18/06/04-18:00</Time_Period>
    </Validity_Period>
    <Condition>
        <Condition_Object>
            <Simple_Variable>
                <Variable_Id> setEmergencyOn </Variable_Id>
                <Variable_Type>localVariable</Variable_Type>
                <Syntax>String</Syntax>
                <Monitoring_Component>
                    <Component_Name>Policy Manager</Component_Name>
                </Monitoring_Component>
            </Simple_Variable>
        </Condition_Object>
        <Condition_Requirement>
            <Requirement_Id> setEmergencyOn </Requirement_Id>
            <Requirement_Type>Max_Threshold</Requirement_Type>
            <Requirement_Parameter>
                <Name>new_sip</Name>
                <Syntax>String</Syntax>
                <Value> 3</Value>
            </Requirement_Parameter>
```

```
        <Target_Condition_Object>
            <Simple_Variable_Id> setEmergencyOn </Simple_Variable_Id>
        </Target_Condition_Object>
        <Req_Evaluator_Component>
            <Component_Name>PolicyManager</Component_Name>
        </Req_Evaluator_Component>
    </Condition_Requirement>
    <Condition_Evaluation_Parameters>
        <Condition_Evaluation_Method>AuthUsers </Condition_Evaluation_Method>
        <Condition_Evaluation_Stop>1</Condition_Evaluation_Stop>
    </Condition_Evaluation_Parameters>
</Condition>
        <PolicyVariable>
        <ContextVariable>
                <ContextEntity>
                        <Person>
                                <name>911</name>
                                <username>911@10.0.1.34</username>
                        </Person>
                        <Person>
                                <name>doctor</name>
                                <username>doc@10.0.1.36</username>
                        </Person>
                        <Person>
                                <name>paramedic</name>
                                <username>paramedic@10.0.1.37</username>
                        </Person>


                </ContextEntity>
                <Activity>InRange</Activity>
        </ContextVariable>
 </PolicyVariable>

<Action>
    <Action_Id>AuthUsers</Action_Id>
    <Action_Type>userAuth</Action_Type>
    <Action_Parameter>
        <Name>userAuth1</Name>
        <Syntax>String[]</Syntax>
        <Value>true</Value>
    </Action_Parameter>
    <Enforcer_Component>
        <Component_Name>SIPBroker</Component_Name>
    </Enforcer_Component>
    <Enforcement_TimeOut>1min</Enforcement_TimeOut>
    <Success_Output_Parameters>
        <Action_Result_Value>success</Action_Result_Value>
```

```
        </Success_Output_Parameters>
    </Action>

</Policy>
```

A.5 Service Supporting Components & Testbeds

A5.1 WLAN Broker

- public boolean isUserAssociated(String ClientIPAddr): returns true if the client identified by the ClientIPAddr is associated to the WLAN AP. A client is associated to an AP when the first two layers (PHY and MAC) are currently active.
- public String[] getAssociatedUserAddrs(): returns the MAC Addresses of all the clients that are associated to the WLAN Access Point.
- public String getUserMAC (String ClientIPAddr): returns the MAC address of the client connected to the WLAN AP with the IP Address specified by ClientIPAddr.
- public String getUserIP (String ClientMACAddr): returns the IP Address of the client connected to the WLAN AP with the MAC Address specified by ClientMACAddr. The MAC Address should be expressed in the following format: ffff.ffff.ffff and the letters should be lowercase.
- public boolean createSS(String SSID, String AutType, String AccessListID, String MaxAssoc): creates a new Service Set on the WLAN Access Point. It returns 'true' if the action is executed and 'false' in any other case. In order to create a Service Set, the following parameters are needed:
    - String SSID: Identification of the Service Set.
    - String AuthType: The type of authentication <<empty>,< Mac-Address>>. If it is empty, there will be no authentication. empty means an empty String (""). In case of mac-address, the clients are going to be authenticated with their MAC Addresses in the AP, when associating to it. The format of the MAC Address will be the same as in the previous methods.

- String AccessListID: The Id of the Access List that is going to use the Service Set. It is not needed if the AuthType is 'empty'. empty has the same meaning than in the previous parameter. The Access List is a list of mac-addresses related to permission or denegation to associate it to the AP.
- String MaxAssoc: Maximum number of clients that can be associated to a Service Set at the same time.

- public boolean removeSS(String SSID): removes an existing Service Set, which is identified by the SSID. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public String getClientSSID(String ClientMACAddr): will return the SS to which a given user, who is identified by its MAC Address, is associated. The format of the MAC Address will be the same as in the previous methods.
- public boolean establishVLAN (String VLANId, String SSID): establishes a VLAN for clients accessing through the WLAN Access Point. It returns 'true' if the action finishes in a correct way and 'false' in any other case. The parameters needed are:
  - String VLANId: The identification of the VLAN.
  - String SSID: Identification of the Service Set related to the VLAN, as far as every VLAN has to be associated to a Service Set.
  - public boolean removeVLAN (String VLANId, String SSID): removes an existent VLAN. The parameters are the same as in the previous method. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public boolean addAccessList (String AccessListID, String permition, String MACAddr): creates a new access list or adds a new client to an existing Access List. It returns 'true' if the action finishes correctly and 'false' in any other case. The parameters are:

- String AccessListID: Identification of the Access List. It must be a number between 700 and 799.
- String Permition: The kind of permission associated to the MAC Address. It can be permit or deny.
- String MACAddr: Client MAC Address that we want to add to the Access List. The format of the MAC Address will be the same as in the previous methods.

- public String[] getAccessList (): gets all the Access Lists that are defined in the Access Point.
- public boolean removeAccessList (String AccessListID): remove the Access List identified by AccessListID. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public String snmpGet (String community, String host, String oid): makes a get of SNMP from the specific host and the provided oids. The parameters are:
  - String community: Community of the SNMP. It can be public.
  - String host: IP Address of the AP from we want to retrieve information.
  - String [] oids: an array of strings containing all the oids that we want to consult.
- public float getLoad(): This method returns the bytes per second of the radio interface of the Access Point from the AP to the clients (the radio downstream). This load is going to be measured in kbps.
- public float getLoadPerConnectionfromAP (String ClientMACAddr): this method returns the bytes per second [kbps] of the traffic from the AP to the Client.
- public float getLoadPerConnectionfromClient (String ClientMACAddr): this method returns the bytes per second [kbps] of the traffic from the Client to the AP.

- public int getSignalQuality (String ClientMACAddr): gets the quality of the signal received from the client identified by ClientMACAddr. The format of the MAC Address will be the same as in the previous methods. This quality is measured from 0 to 100 in %.
- public int getInterfaceStatus (int ifNum): gets the status of the interface determined by ifNum. It can be 1 if up or 0 if down. If ifNum is '0' it refers to the FastEthernet interface, if it is '1' it refers to the 802.11b interface and '2' refers to the 802.11a interface.
- public boolean setInterfaceStatus (int ifNum, boolean status): sets the status determined on the heading of the radio interface, that is also determined on the heading, following the instructions of the previous method. It returns 'true' if the action is executed and 'false' in any other case.
- public boolean setChannel(String ChannelNumber, int ifNum): set the channel (frequency) that is going to be used by the AP. In the case of 802.11b ChannelNumber must be a number between 1 and 13. In the case of 802.11a ChannelNumber must be one of the following list: 36, 40, 44, 48, 52, 56, 60 or 64. The 'ifNum' follows the coding described on the previous method. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public boolean isChannelActive(int ChannelNumber, int ifNum): checks if the channel specified in the heading is active. In other words, if it is the channel that it is in use at the moment. In the case of 802.11b ChannelNumber must be a number between 1 and 13. In the case of 802.11a ChannelNumber must be one of the following list: 36, 40, 44, 48, 52, 56, 60 or 64. The 'ifNum' follows the coding described on the previous method. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public boolean setMACFilter (String FilterName, String[] MACAddress, String AccessListID): establish a Filter that is going to be used to select the client's MACAddress for QoS purposes. It creates a new user access list for the user

selected or adds the user to an existing access list. It returns 'true' if the action finishes in a correct way and 'false' in any other case. The parameters are:

  - String FilterName: The name that we want to give to the filter.
  - String[] MACAddress: List of Clients' MAC Addresses that we want to include in the filter. The format of the MAC Address will be the same as in the previous methods.
  - String AccessListID: Identification of the Access List. It must be a number between 700 and 799.

- public boolean setVLANFilter (String FilterName, String VLANId): establish a Filter that is going to be used to select the VLAN for QoS purposes. It returns 'true' if the action finishes in a correct way and 'false' in any other case. The parameters are:
  - String FilterName: The name we want to give to the filter.
  - String VLANId: Identification of the VLAN that we want to assign QoS.
- public boolean removeFilter(String FilterName): removes a filter created using one of the two last methods. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public boolean setQoSPolicy (String PolicyName, String FilterName, String Priority): sets a policy to provide QoS. It returns 'true' if the action finishes in a correct way and 'false' in any other case. The parameters are:
  - String PolicyName: The name that we want to give to the policy.
  - String FilterName: The name of an already existing filter, that we want to bind to the policy.
  - String Piority: Which priority level (0...7) we want to assign to this policy: '0' is the lowest and '7' the highest.

- public boolean removeQoSPolicy (String PolicyName): removes a policy created by the previous method. It returns 'true' if the action finishes in a correct way and 'false' in any other case.
- public boolean bindQoS (int ifNum, String direction, String PolicyName): binds a policy, created by the 'setQoSPolicy' method with an Interface and a direction on the flow of data. It returns 'true' if the action finishes in a correct way and 'false' in any other case. The parameters are:
    - o int ifNum: Which interface we want to bind to the policy defined in the previous method. '0' refers to Fast Ethernet, '1' to 802.11b and '2' to 802.11a interfaces respectively.
    - o String direction: In which direction we want to apply the policy. 'input' if it is the direction to the AP, 'output' if it is the direction from the AP and 'both' if we want to apply it to both directions.
    - o String PolicyName: Name of the police, that already exists, which define the QoS.
- public boolean unbindQoS (int ifNum, String direction, String PolicyName): undoes the action made by the previous method, i.e., removes QoS from a specific interface and direction. The parameters have the same meaning as in the previous method. It returns 'true' if the action finishes in a correct way and 'false' in any other case.

Finally, the WLAN Broker interface adopts the following structure:

```
public class WLANBrokerInterface
{
public WLANBrokerInterface(InetAddr APAddr, String confPasswd);
public boolean isUserAssociated(String ClientIPAddr);
public String[] getAssociatedUserAddrs();
public String getUserMAC (String ClientIPAddr);
public String getUserIP (String ClientMACAddr);
```

```
public boolean createSS(String SSID, String AutType, String AccessListID, String
MaxAssoc);
public boolean removeSS(String SSID);
public String getClientSSID(String ClientMACAddr);
public boolean establishVLAN (String VLANId, String SSID);
public boolean removeVLAN (String VLANId, String SSID);
public boolean addAccessList (String AccessListID, String permition, String MACAddr);
public String[] getAccessList ();
public boolean removeAccessList (String AccessListID);
public String snmpGet (String community, String host, String oid);
public float getLoad();
public float getLoadPerConnectionfromAP(String ClientMACAddr)
public float getLoadPerConnectionfromClient (String ClientMACAddr);
public int getSignalQuality (String MACAddress);
public int getInterfaceStatus (int ifNum);
public boolean setInterfaceStatus (int ifNum, boolean status);
public boolean setChannel(String ChannelNumber, int ifNum);
public boolean isChannelActive(int ChannelNumber, int ifNum);
public boolean setMACFilter (String FilterName, String[] MACAddress, String
AccessListID);
public boolean setVLANFilter (String FilterName, String VLANId);
public boolean removeFilter (String FilterName);
public boolean setQoSPolicy (String PolicyName, String FilterName, String Priority);
public boolean removeQoSPolicy (String PolicyName);
public boolean bindQoS (int ifNum, String direction, String PolicyName);
public boolean unbindQoS (int ifNum, String direction, String PolicyName);
}
```

A5.2 IP-GRE Tunnel Set-up

Setting up the IP-GRE tunnels, the Linux iproute2 [105] package was used. The tunnel configuration was done as follows:

```
iface ucl0 inet static
      address 127.0.1.20
      netmask 255.255.255.255
      pre-up ip tunnel add ucl0 mode gre local 128.40.42.180 remote
130.188.225.122 dev eth0
      up ip link set ucl0 multicast off
      post-down ip tunnel del ucl0
```

and the reverse path:

```
iface ucl1 inet static
      address 127.0.1.21
      netmask 255.255.255.255
      pre-up ip tunnel add ucl1 mode gre local 130.188.225.122 remote
128.40.42.180 dev eth0
      up ip link set ucl1 multicast off
      post-down ip tunnel del ucl1
```

A5.3 The SIP Broker

The WLAN Broker was used in the Emergency Support Service Scenario. The methods offered by is interface are:

public int userStatus(SIPUrl SIPUserAddr): returns the status of a SIP user, whether he is disconnected, in an active session or in an inactive session

public InetAddress getUserIP(SIPUrl SIPUserAddr): returns the IP address of the host that the SIP user is logged into during the SIP session given his SIP address.

public int maxSessions(): returns the maximum number of SIP sessions, which is allowed by the SIP softswitch.

public int totalSessions(): returns the number of current SIP sessions currently handled by the SIP softswitch. This is a rough measure of the traffic through the system.

public String[] getProxyServers(): returns an array of proxy servers found in the domains that the SIP softswitch serves.

public String[] sessionStateInfo(String sessionID) and public Hashtable sessionStateInfo(String SIPProxyID): returns information regarding ongoing SIP sessions. The information contains the state of the SIP session, the caller and the callee, session id, media description etc. The first method returns the status of a particular session, given the session ID while the other method returns a hash table of all the session IDs and the status of their sessions within a particular domain.

public boolean terminateSession(String sessionID, SIPUrl CallerAddr, SIPUrl CalleeAddr): terminates a session according to the sessionID, the initiator (caller) and the recipient (callee).

public boolean terminateAllSessions(String SIPProxyID): terminates all the ongoing sessions in that are in the domain of a certain SIP proxy server.

public void setCallAcceptanceDiverter (String SIPProxyID SIPCallDiverter diverter): this method sets the particular SIPCallDiverter. The SLO instantiates this SIPCallDiverter object and passes it to the SIP broker.

public void applyDiverter (String SIPProxyID boolean apply): this method is invoked by the SLO when a crisis condition occurs. Then all call admission tasks go through the diverter.

The interface adopts the following structure:

```
public class SipBrokerInterface {
//User status
public static int USER_ACTIVE = 0;
public static int USER_INACTIVE = 1;
public static int USER_DISCONNECTED = 2;

public SipBrokerInterface();
public int userStatus(SIPUrl sipUserAddr);
public InetAddress getUserIP(SIPUrl SIPUserAddr);
public int maxSessions();
```

```
public int totalSessions();

public String[] getProxyServers();
public String[] sessionStateInfo(String sessionID);
public Hashtable sessionStateInfo(String SIPProxyID);
public boolean terminateSession(String sessionID, SIPUrl CallerAddr, SIPUrl
CalleeAddr);
public boolean terminateAllSessions(String SIPProxyID);
public void setCallAcceptanceDiverter (String SIPProxyID SIPCallDiverter diverter);
public void applyDiverter (String SIPProxyID boolean apply);
}
```

A5.3.1 SIP Broker Set Up

- The code for the Sip Broker is made up of three parts. The code for the Sip Broker, the code for the Sip Broker interface and the Siptrex platform code.
- The original Siptrex program was written by students at the Computer Science Department at UCL and can be found at www.siptrex.net.
- Some classes in the net/siptrex/callapiuser/useragent were changed. These classes are the UserAgent.java, GuiListener.java and Gui.java. For the user agent to communicate with the SIP broker a new package was created called net/siptrex/kerry which contained the AgentInterface.java, Agent.Exception.java and ByeListenerServer.java.
- The files in the sessionSupport and sipBroker directory have to be put in the Dina directory on the Linux machine. The files in the sessionSupport directory go into the sessionSupport folder in the Dina directory and the sipBroker folder is put in the Dina home.
- The whole thing runs like this. The Sip Broker is run as an application from the Linux box which has Dina using java sipBroker.SipBroker. The SipBrokerInterface class is in the sessionSupport folder in the Dina distribution and is called by active entities. The SipBrokerInterface starts up a SBIntServer The SipBrokerInterface accesses the SBIntServer through the SBInterface class.

- There are two hash tables in the Sip Broker, one for keeping track of the state of the SIP sessions, with the key as the sessionID and the object a space separated string containing the caller, callee and state. The state is given as "0" for a pending call and "1" for an active call.
- When a user agent makes a call, it starts a ByeListenerServer to check if the SIP broker wants to stop the call in the event of a crisis occurring. The batch file which starts up the user agent has to be edited to include the IP address of the SIP broker. This IP comes last after the entry for the listening point file.

A.6 Siptrex Tutorial

A6.1 Introduction

This tutorial will guide you through the steps required to get the Siptrex user agent and servers up and running. This comes together with the siptrex platform.

The first step is to unzip the distribution zip into a directory on your local machine. For the purposes of this tutorial, we will call this directory [path to Siptrex]. This tutorial assumes you have Java 2 Platform, SE v1.3 running on your system. All components in the system will provide their usage if called with no command line parameters.

A6.2 Location server

The first component that you should configure and instantiate is the location server. This is a proprietary Siptrex server that uses JDBC and a database to provide location services for SIP request routing and presence services. A batch file that starts the server (location.bat) is included in the root of the Siptrex distribution. The usage for the location server is as follows:

Usage: java net.siptrex.location.LocationServer <TCP Port>

The TCP port is the local port on which the server listens. The location server will listen on the default IP address for you system. The location server requires an ODBC data source named "siptrex" to be available. An example Microsoft Access database is included in the Siptrex distribution here:

[path to Siptrex] /net/Siptrex/location/location.mdb.

A6.3 Feature Server

The feature server is required by the user agent to provide contact and presence information about other Siptrex users. A batch file is included to start the feature server

(feature.bat). You should edit this batch file to give the correct parameters for your system. The usage for the feature server is:

Usage: java net.siptrex.feature.FeatureServer <TCP Port> <Location Server IP> <Location Server port> <Local Domain> [<user file>]

The location server IP address and port should correspond to those for your system as configured above. The local domain parameter specifies the local domain of this feature server (i.e. that for which it has responsibility). This is used for registration and routing purposes. The user file parameter allows an existing set of users to be loaded from disk. Running the feature server without specifying the <user file> parameter will cause the command line interface to be provided. This provides the following commands for user management: <open> <save> <add> <remove> <print> <help> <quit>.

A6.4 User agent

The user agent requires that you have the Siptrex Feature up and running, as described above. A batch file is included to start the user agent. You should edit this file to give the correct parameters for your system (useragent.bat). The usage for the user agent is:

Usage: java net.siptrex.callapiuser.useragent.Gui <Proxy IP> <Proxy Port> <Feature Server IP> <Feature Server Port> <Listening Points File> <SIP broker IP>

The user agent assumes you are running an outbound proxy. Follow the instructions below to install and configure the Siptrex proxy. The Feature Server details should correspond to those for your system, as configured above. The listening points file provides a list of listening points (IP address and port tuples) to be used by the user agent Siptrex SIP stack. An example listening points file is included in the Siptrex distribution here: [path to Siptrex]/inifiles/useragent/listeningpoints.ini.

NB. When closing the user agent use the exit button rather than the usual x button common to most windows.

A6.5 Registrar

The registrar provides the facility for a user agent to register a user as being present at a particular location. A batch file is provided in the root of the Siptrex distribution which should be edited to suit your system configuration. The usage for the location server is as follows:

java net.siptrex.jainuser.registrar.Registrar <Location Server IP> <Location Server Port> <Listening Points File>.

The location server details should be specified as configured above. The listening points file contains a list of listening points to be used for the registrars Siptrex SIP stack. The first of these IP-port tuples that is available, will be used for this instance of the Siptrex SIP stack. An example listening points file is included in the Siptrex distribution here: [path to Siptrex]/inifiles/registrar/listeningpoints.ini.

A6.6 Proxy

The Siptrex proxy takes a single parameter - namely the configuration file to be used for the proxy. The configuration file should contain a list of parameters of the following form:

[local_domain]	ucl.fa
[local_listening_point_ini_file]	inifiles\proxy\listeningpoints.ini
[non-local_domain_proxy_mappings]	inifiles\proxy\domainmap.ini
[location_server_ip]	192.168.0.1
[location_server_port]	7000
[registrar_ip]	192.168.0.1
[registrar_port]	8000
[loadbalance_inifile]	inifiles\proxy\loadbalance.ini

The local domain is used for routing decisions (i.e. whether to check the location server for location information). The listening point's ini file contains a list of IP addresses and

port numbers that this proxy could listen on. The highest of these on the list, that is available, will be used. The domain map file contains a list of domain to IP mappings for use when resolving non local domains. The location server IP and port should correspond to those configured above. The registrar IP and port should also correspond to those defined in the listening points ini file for the registrar above. The load balance ini file specifies properties as used by the load balancing routing strategy. Example proxy initialisation, listening point, domain map, and load balance files are included in the Siptrex distribution here: [path to Siptrex]/inifiles/proxy/.

A.7 Glossary

Active Technology: The technology that enables code migration. In this thesis, both mobile agent technology and active/programmable networks technology are regarded as active technology.

Ad-hoc network: An Ad-hoc network is a network, which is established dynamically by (mobile) devices and is maintained by them for their communication needs.
ASN.1 In telecommunications and computer networking Abstract Syntax Notation one (ASN.1) is a standard and flexible notation that describes data structures for representing, encoding, transmitting, and decoding data. It provides a set of formal rules for describing the structure of objects that are independent of machine-specific encoding techniques and is a precise, formal notation that removes ambiguities. A free e-tutorial is available online to check your understanding of the ASN.1 syntax and semantics.

ATM: Asynchronous Transfer Mode, or ATM for short, is a cell relay network protocol which encodes data traffic into small fixed sized (53 byte; 48 bytes of data and 5 bytes of header information) cells instead of variable sized packets as in packet-switched networks (such as the Internet Protocol or Ethernet).

Brokers: The brokers are modules that give active services the capability to utilise host information and resources and perform operations in the local environment.
CAPL short for Context-Aware Policy Language, is a kind of rule-based language used for network and service management with strong context-awareness support.

CAPL Decision Maker: The CAPL Decision Maker (CAPLDM) receives the policy conditions from the Policy Manager and is responsible for the condition evaluation process. Once this is successful a notification is sent to the Policy Manager announcing that the condition was met.

CAPL Policy Enforcement Points: The Policy Enforcement Points are the responsible for implementing the policy actions.

CAPL Policy Management Tool: The Policy Management Tool (PMT) is the user interface for inserting policies.

CAPL Policy Manager: This component is responsible deciding which policies are applicable when a set of given conditions are met.

CAPL Policy Storage Service: The Policy Storage Service (PSS), is the place where all the policies are stored.

Common Information Model: The Common Information Model (CIM) is a management schema, which consists of an object-oriented model for the representation of the information that will be stored in the directory of a DEN -enabled network

Context: Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

CONTEXT: EU IST Project CONTEXT stands for "Active Creation, Delivery and Management of Efficient Context Aware Services". It was a medium-sized project including eight partners from Europe. It started on September 2002 and lasted for 2.5 years. Its main objective was to specify and design models and solutions for an efficient provisioning of context-based services making use of active networks on top of fixed and mobile infrastructure.

Context awareness: Context-awareness is the abilitity to use context information

DINA: Stands for DINA Is Not ABLE.An active platform which was selected to be used for distributing the policies and monitoring the different conditions set by the policies.

Directory Service: The term directory service means the collection of software, hardware, and processes that store information needed and make that information available to users or applications. A directory service consists of at least one instance of Directory Server and one or more directory client programs. One common directory service example is a Domain Name System (DNS) server.

INMS: An Inter-technology domain Network Management System (INMS) developed by the WINMAN project as a sub-layer of the Network Management Layer was implemented to support IP-connectivity spanning different WDM sub-networks and integrate the management of IP and WDM transport networks. The INMS, the IP and the WDM Network Management Systems implement Configuration, Fault and Performance (CFP) management application functions.

IP: The Internet Protocol (IP) is a data-oriented protocol used by source and destination hosts for communicating data across a packet-switched internetwork.Data in an IP internetwork are sent in blocks referred to as packets or datagrams (the terms are basically synonymous in IP). In particular, in IP no setup is needed before a host tries to send packets to a host it has previously not communicated with.

Iptables: iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains. Each chain is a list of rules, which can match a set of packets. A rule consists of a specific match criteria and target. Target tells what to do matched packets, which may be a jump to a user-defined chain in the same table or drop packet

ISO OSI Management Architecture: ISO (International Standard Organization) OSI (Open System Interconnection) management architecture, as part of the whole OSI standard suite, is regarded as the first standard aiming for network management. It significantly influenced the basic framework of latter network management standards

such as ITU-T TMN and TMF, etc. The core of its architecture is so called FCAPS, i.e. the management of fault, configuration, accounting, performance and security

IST: One of the thematic priorities in the European Union Sixth Framework Programme for research and technological development set for the period 2002-2006.The research activities funded in this framework address both technical aspects (hardware, software, knowledge) and societal challenges (communication, collaborative working, education) of Information Technologies.

LDAP: Lightweight Directory Access Protocol (LDAP) is a protocol for accessing on-line directory services.

Linux: Linux is a computer operating system and its kernel. It is one of the most famous examples of free software and of open-source development: unlike other major operating systems (such as Windows or Mac OS), its underlying source code is available to the public and anyone can freely use, modify, and redistribute it.

MIB: A management information base (MIB) comprises a collection of objects in a (virtual) database used to manage entities (such as routers and switches) in a network. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1), the software that performs the parsing is a MIB compiler. The database is hierarchical (tree structured) and entries are addressed through object identifiers. Internet documentation RFCs discuss MIBs, notably RFC 1065, "Structure and Identification of Management Information for TCP/IP based internets", and its two companions, RFC 1066, "Management Information Base for Network Management of TCP/IP-based internets", and RFC 1067, "A Simple Network Management Protocol".

Mobile Agent: In computer science, a mobile agent is a piece of computer software that is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer.

MPLS: In computer networking and telecommunications, Multiprotocol Label Switching (MPLS) is a data-carrying mechanism, operating at a layer below protocols such as IP. It was designed to provide a unified data-carrying service for both circuit-based clients and packet-switching clients which provide a datagram service model. It can be used to carry many different kinds of traffic, including both voice telephone traffic and IP packets.

Netfilter: netfilter is a set of hooks inside the Linux kernel's network stack, which allows kernel modules to register call-back functions called every time a network packet traverses one of those hooks

Policy Decision Point: The policy consumer is the component that retrieves policies from repository, evaluates them and sends the necessary commands to the policy target. Of course the evaluation of certain policies may also be done on the policy target, if it is capable of understanding policies.

Policy Definition Language: Policy Definition Language (PDL) is designed for satisfying the need of mapping requirements for services to be provided by the network as defined in a business specification (e.g., an SLA) to a common vendor- and device-independent intermediate form.

Policy Defintion Notation: This approach is based on the ODP distributed computing platform, and the authors choose the area of performance management for the application of policies. Their notation looks very much like a programming language

Policy Enforcement Point: The policy target is the managed device, where the policy is finally enforced.

Policy Framework Definition Language: The Policy Framework Definition Language (PFDL) proposed by IETF was supposed to satisfy the objectives of a PDL as mentioned in the very beginning of this sub-section. But due to some unclear reason, the IETF policy group has decided to suspend work on the policy definition language.

Policy Information Model: IETF PCIM presents the object-oriented information model for representing policy information developed jointly in the IETF Policy Framework Working Group and as extensions to the Common Information Model (CIM) activity in the Distributed Management Task Force (DMTF). This model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and association classes that indicate how instances of the structural classes are related to each other

Policy Management Tool: With this tool new policies in a policy based management system can be defined, existing ones edited, or simply viewed.

Policy Repository: The policy repository is used for the storage of policies, after they have been defined and validated by the policy management tool. There should also exist a protocol that enables read and write access to the repository. The general framework does not require a specific implementation for the policy repository, or the repository access protocol.

Policy Template Definition: In this approach, apart from the policy classification and the policy hierarchy, R. Weis also derived a policy template definition, which is then transformed into policy objects/management scripts.

Policy-based context information model: The policy-based context information model is designed based on the IETF PCIM (Policy Core Information Model) and its extension, directions which are followed by most of the work carried out in the policy-based management field in order to model context aware policies

Policy-based Management: a relatively new network (and service) management method that uses policies to control the behaviours of management system. Policy is in the form of IF <condition(s)> THEN <action(s)>. The core of policy-based management is a trio: policy definition language, policy information model and policy decision making engine.

Ponder: Ponder is a language developed by the Policy Research Group in Imperial College London for specifying management and security policies for distributed systems and network management systems. Ponder supports authorisation, filter, refrain and delegation policies for specifying access control and obligation policies to specify management actions.

RFC: A Request for Comments (RFC) document is one of a series of numbered Internet informational documents and standards very widely followed by both commercial software and freeware in the Internet and Unix communities. They are now published under the aegis of the Internet Society (ISOC, an open organization whose mission is developing the Internet for the benefit of people throughout the world) and its technical standards-setting bodies.

SDH: Synchronous Optical Networking, commonly known as SONET, is a standard for communicating digital information over optical fiber. It was developed to replace the PDH system for transporting large amounts of telephone and data traffic. It is defined by GR-253-CORE from Telcordia.The more recent Synchronous Digital Hierarchy (SDH) standard is built on experience in the development of SONET. Both SDH and SONET are widely used today; SONET in the U.S. and Canada, SDH in the rest of the world. SDH is growing in popularity and is currently the main concern with SONET now being considered as the variation.

Service Level Agreement: A Service Level Agreement (SLA) is a formal written agreement made between two parties: the service provider and the service recipient. The SLA itself defines the basis of understanding between the two parties for delivery of the service itself. The document can be quite complex, and sometimes underpins a formal contract. The contents will vary according to the nature of the service itself, but usually includes a number of core elements, or clauses.

Siptrex: The Siptrex system, as its name suggests, implements the functions of an IP Centrex.

SNMP: The Simple Network Management Protocol (SNMP) forms part of the internet protocol suite as defined by the Internet Engineering Task Force. The protocol can support monitoring of network-attached devices for any conditions that warrant administrative attention.

TCP: The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite. Using TCP, programs on networked computers can create connections to one another, over which they can send data. The protocol guarantees that data sent by one endpoint will be received in the same order by the other, and without any pieces missing. It also distinguishes data for different applications (such as a Web server and an email server) on the same computer.

TELNET: It is a network protocol. It is typically used to provide user oriented command line login sessions between hosts on the Internet. The name is derived from the words terminal emulation, since the program is designed to emulate a single terminal attached to the other computer.

TINA-C: The intention of TINA Consortium [TINA-web] is to define a common architecture upon which next generation telecommunication services could be built. Same as the principle proposed in TMN, the architecture proposes logically separated high level applications from the physical infrastructure (e.g. the network resources). Its information model, as a central part of its architecture, is fully OO based and furthermore, a specific OO DPE (Distributed Programming Environment) is used, i.e., CORBA (Common Object Request Broker Architecture) from OMG (Object Management Group).

TMF: The Tele-Management Forum (TMF - formerly NMF) can be seen as a step further to TMN in the sense that it aims to identify, create, develop, and implement real

world solutions that automate and streamline telecom operations. The TMF always takes a pragmatic approach in its conclusions: real world and product solutions are sought, as specifically claimed in the introduction of the Catalyst projects. This entails the use of already existing and quite mature technologies and products.

TMN: The management architecture of the ITU-T is known as the "Telecommunications Management Network" (TMN). TMN consists of several smaller architectures: a functional architecture, a physical architecture, an information architecture (including many ideas of ISO management), and a logical layered architecture (with a responsibility model). The TMN methodology makes good use of the OSI systems management principles and is sometimes considered as an application of OSI principles. TMN architecture is also based on an object-oriented approach and its information model is defined using GDMO template and ASN.1 in ITU-T standards.

UDP: The User Datagram Protocol (UDP) is one of the core protocols of the Internet protocol suite. Using UDP, programs on networked computers can send short messages known as datagrams to one another. UDP does not provide the reliability and ordering guarantees that TCP does; datagrams may arrive out of order or go missing without notice. However, as a result, UDP is faster and more efficient for many lightweight or time-sensitive purposes.

UNIX: Unix or UNIX is a computer operating system originally developed in the 1960s and 1970s by a group of AT&T Bell Labs employees including Ken Thompson, Dennis Ritchie, and Douglas McIlroy. Today's Unix systems are split into various branches, developed over time by AT&T, several other commercial vendors, as well as several non-profit organizations.

VPN: A Virtual Private Network, or VPN, is a private communications network usually used within a company, or by several different companies or organizations, communicating over a public network. VPN message traffic is carried on public networking infrastructure (e.g. the Internet) using standard (often insecure) protocols.

WDM: In telecommunications wavelength-division multiplexing (WDM) is a technology which multiplexes several optical carrier signals on a single optical fibre by using different wavelengths (colours) of laser light to carry different signals. This allows for a n-fold increase in capacity, in addition to making it possible to perform bidirectional communications over one strand of fibre.

Web Service Technology: Web service is a standards-based, service-oriented technology that integrates PCs, other devices, databases, and networks into one virtual computing fabric to be used by users via browsers. The services themselves usually run on Web-based servers (instead of PCs), therefore moving functions from the desktop to the Internet. Cross-platform capabilities (or interoperability) are one of Web service's key attractions.

WINMAN: WINMAN was a European research and development project, which ran from July 2000 to April 2003. It aimed to offer an integrated network management solution for the provisioning and maintenance of IP over WDM end-to-end transport services derived from Service Level Agreements (SLAs).

XML: The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language for creating special-purpose markup languages. It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet. Languages based on XML (for example, RDF, RSS, MathML, XHTML and SVG) are themselves described in a formal way, allowing some programs to modify and validate documents in these languages without prior knowledge of their form.

A.8 List of Publications

This section contains papers and book chapters by the author:

- Karayannis, F., Serrat, J., Baliosian, J., Rubio. J., Vaxevanakis, K., Pagomenos, G., Zahariadis, T., Raptis, L., Ellinas, M., Chatzilias, G., Chronis D., Katopodis, H., Doukoglou, T., Androulidakis, S., Galis, A., Vardalachos, N., "In Field evaluation of a managed IP/MPLS over WDM provisioning solution", Accepted for publication in IEEE Communications Magazine
- Jean, K., Vardalachos N., Galis, A., "Context-aware VoIP Services", submitted to the 2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005), Montreal, Canada, 17-19 October 2005
- Galis, A., Serrat, J. Ras, D., Juhola, A., Georgatsos, P., Serrano, J.M., Justo, J., Marin, R., Cohen, R., Ahola, K., Vardalachos, N., Jean, K., Damilatis, T., "ContextWare Programmable Middleware", 2nd International Workshop on Managing Ubiquitous Communications and Services, MUCS 2004, Dublin, Ireland, 13-14 December 2004
- Yang, K., Galis, A., Serrat, J., Jean, K., Vardalachos, N., Guo, X.,"Network-Centric Context-aware Service over WLAN and GPRS Networks", proceedings of the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Beijing, China, 7-10 September 2003
- Vardalachos, N., Rubio, J., Galis, A., Serrat J., "A Policy Management System for Hybrid Networks", proceedings of the London Communications Symposium, University College London, London, United Kingdom, 8-9 September 2003
- Vardalachos, N., Grampin, E., Galis, A., Serrat J., "A Policy Manager for IP over WDM Networks", to be presented at Eurescom 2002; Powerful Networks for Profitable Services, Heidelberg, Germany, 21-24 October 2002
- Vardalachos, N., Grampin, E., Galis, A., Serrat J., "Using Policies on Management of Hybrid Networks", proceedings of the London Communications Symposium, University College London, London, United Kingdom, 9-10 September 2002
- Grampin, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, A., "Implementation Issues in PBNM Systems", paper published in the magazine Hiradastechnika (in Hungarian).
- Vardalachos, N., Grampin, E., Galis, A., Serrat J., "Policy Management Approach for IP over WDM Networks: A Synthesis Study", proceedings of the 6th London Communications Symposium, University College London, 10-11 September 2001

- Grampin, E., Rubio, J., Vardalachos, N., Galis, A., Serrat, J., "Implementation issues of policy based network management systems", to be presented on the 3rd International Workshop on Design of Reliable Communication Networks (DCRN2001), Budapest, Hungary, 7-10 October 2001
- Garcia, R., Vardalachos, N. Grampin, E., Raptis, L., Karayannis, F., Vivero, J., Serrat, J., "An approach on policy-based management for IP connectivity over WDM networks", 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Orlando, USA, 22-25 July 2001
- Raptis, L., Karayannis, F., Serrat, J., Vaxevanakis, K., Galis, A., Arozarena, P., Vardalachos, N., Chatzilias, G., Chronis, D., Garcia, R., Romijn, W., Philipopoulos, P., Patikis, Y., Josef, D., Schwartz, A., Zahariadis, T., "Integrated Management of IP over Optical Transport Networks", proceedings of IEEE International Conference on Telecommunications, Bucharest, Romania, 4-7 June 2001
- Vardalachos, N., Galis, A., Serrat, J., Garcia, R., Hoekstra, G., "Policy Based Management for IP over WDM Networks", proceedings of IEEE International Conference on Telecommunications, Bucharest, Romania, 4-7 June 2001
- Vardalachos, N., "Bandwidth Management for IP QoS", proceedings of the 5th London Communications Symposium, University College London, 14-15 September 2000
- Also contributed in several chapters of the book "Deploying and Managing IP over WDM networks", by J. Serrat and A. Galis, Publishers, ISBN:1-58053-501-1, Boston 2003, Arthech House